Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Peter Rot

Razvoj s certifikati varovane spletne trgovine

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Aleš Smrdel

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom LaTeX.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Peter Rot sem avtor diplomskega dela z naslovom 'Izdelava s certifikati varovane spletne trgovine'. S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Aleša Smrdela,
- so elektronska oblika diplomskega dela, naslov (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 20.8.2015

Podpis avtorja:

"Tisto, kar si um lahko zamisli in v kar verjame, lahko tudi doseže."

W. Clement Stone

Rad bi se zahvalil očetu Hubertu, ki me je učil marsikatere življenjske modrosti, in mami Anamariji za brezpogojno ljubezen. Iskrena hvala sestri Zali in Tadeju, ki sta mi v času študija s svojo razigranostjo polepšala konce tednov. Hvala dekletu Martini, ker je potrpežljivo poslušala razlage programerskih rešitev in vsem, ki ste (tako ali drugače) prispevali k nastanku tega dela. Hvala tudi mentorju doc. dr. Alešu Smrdelu za potrpežljivost in usmerjanje pri delu.

Kazalo

Seznam uporabljenih kratic

kratica	angleški pomen	slovenska razlaga
AJAX	Asynchronous Javascript and XML	Asinhroni Javascript in XML
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart	Test, katerega cilj je ugotoviti, ali z računalnikom upravlja človek ali ne
CMS	Content Management System	Sistem za upravljanje vsebin
CSRF	Cross Site Request Forgery	Spletni napad, pri katerem napadalec ponaredi zahtevo in jo na skrit način podtakne avtenticiranemu uporabniku, da jo le-ta izvede na spletni strani, kjer je avtenticiran
CSS	Cascading Style Sheets	Kaskadne slogovne pole - predloge, ki opisujejo vizualno obliko spletne strani
D3.js	Data-Driven Documents	JavaScript knjižnica za manipulacijo dokumentov, ki so osnovani na množici podatkov
DOM	Document Object Model	Dokumentni objektni model - dogovorjen način za prezentacijo HTML in XML dokumentov
DoS	Denial-of-Service attack	Napad za zavrnitev storitve
DRY	Don't Repeat Yourself	Ne ponavljaj se - Princip razvoja programske opreme, s katerim programiramo bolj optimalno
HTML	Hyper Text Markup Language	Označevalni jezik za izdelavo spletnih strani

HTTP	Hypertext Transfer Protocol	Spletni protokol za prenos podatkov
JSON	JavaScript Object Notation	Standardiziran format za prenašanje sporočil
MD5	Message-Digest Algorithm 5	Zgoščevalna funkcija za kriptiranje nizov, katere izhod je dolg 128 bitov
MITM	Man In The Middle	Napad "Človek v sredini"
OWASP	Open Web Application Security Project	Neprofitna organizacija, ki se trudi izboljševati varnost na spletu
PDO	PHP Data Object	PHP programski objekt, namenjen za izvajanje operacij nad podatkovno bazo
PHP	Hypertext Preprocessor	Skriptni jezik, ki se uporablja za izdelavo dinamičnih spletnih strani
REST	Representational State Transfer	Vrsta arhitekturnega stila spletnih aplikacij
ReDoS	Regular Expression DoS	Napad za zavrnitev storitve, ki temelji na dejstvu, da se nekatere regularne izraze dolgo preverja
RSA	Rivest, Shamir, Adelman	Algoritem za šifriranje z javnim ključem
SHA1	Secure Hash Algorithm 1	Zgoščevalna funkcija za kriptiranje nizov, katere izhod je dolg 160 bitov
SQL	Structured Query Language	Programski jezik, s katerim se izvaja operacije nad podatkovno bazo
SSL	Secure Sockets Layer	Kriptografski protokol, namenjen vzpostavljanju varnih komunikacijskih povezav, predhodnik TLS
URL	Uniform Resource Locator	Enolični krajevnik vira
WAMP	Windows, Apache, MySQL, PHP	Skupek programske opreme za popolnoma delujoč spletni strežnik

TLS	Transport Layer Security	Kriptografski protokol, namenjen vzpostavljanju varnih komunikacijskih povezav
WYSIWYG	What You See Is What You Get	Programi, ki omogočajo direkten predogled izdelka
XML	Extensible Markup Language	Razširljiv jezik za označevanje
XSS	Cross Site Scripting	Spletni napad, pri katerem napadalec aplikaciji pripiše kodo, ki se ponavadi izvede pri drugem uporabniku, ki aplikacijo uporablja
ZAP	Zed Attack Proxy	Orodje za iskanje varnostnih lukenj v spletnih aplikacijah

Povzetek

Spletne trgovine vsako leto privabijo več kupcev. Pri programiranju tovrstnih aplikacij moramo biti pozorni na dve stvari: na realizacijo funkcionalnosti ter zagotavljanje varnosti. V tej diplomski nalogi je razvita spletna trgovina s tremi moduli (modul za kupce, prodajalce ter za naročanje pri dobaviteljih). Pri razvoju je poseben poudarek namenjen zagotavljanju varnosti. Pri tem smo upoštevali priporočila organizacije OWASP. Implementirana je obramba proti injekcijam, XSS, CSRF ter proti napadom, ki izkoriščajo napačno upravljanje z avtentikacijo in sejami. Poleg tega so izdelani samopodpisani certifikati X.509, ki so potrebni za prijavo v zaledni sistem.

Ključne besede: spletna trgovina, varnost, OWASP, Apachi, X.509.

Abstract

Each year web stores attract more customers. When programming these types of applications we have to consider two things: implementation of functionality and ensuring web security. A web store with three modules has been developed in the following thesis (a module for buyers, sellers and also for managing supplies). Special emphasis during development is placed on security. We also considered the guidelines from the OWASP organization. A defense against injection, XSS and CSRF has been implemented, as well as against attacks that exploit the broken authentication and session management. In addition, self-signed certificates X.509 were created which are necessary for signing in back-end system.

Keywords: web store, security, OWASP, Apachi, X.509.

Poglavje 1

Uvod

Cilj dela je opisati principe programiranja varnih spletnih aplikacij in realizirati pred vdori (čim bolj) varno spletno trgovino. Tematika je pomembna predvsem zato, ker so tehnologije za izdelavo spletnih storitev postale zelo lahko dostopne in razširjene, sorazmerno s tem pa so se razširile tudi zlorabe. Zato se je potrebno programiranja lotiti na pravi način, da s tem zmanjšamo ranljivosti naših aplikacij. Napadalcu je potrebno za nezaželeno uporabo aplikacije najti zgolj eno pot, da povzroči škodo, zato bi moral razvijalec programa teoretično predvidevati vse možnosti vdora. Moja aplikacija je razvita v tehnologijah PHP, JavaScript, HTML, CSS in SQL na strežniku Apache [1]. Strežnik Apache je svetovno najbolj razširjen [2] odprtokodni HTTP strežnik za moderne operacijske sisteme, kot sta Linux in Windows. Te tehnologije sem izbral, ker se da z njimi programirati na dovolj nizkem nivoju za razumevanje hipotetičnih vdorov. Želim torej izdelati aplikacijo, ki bo poleg tega, da bo imela delujoče funkcionalnosti spletne trgovine, odporna na različne načine napadov, ki jih bom podkrepil s primeri. Uspešnost pri zagotavljanju varnosti bom sproti in na koncu testiral s programom OWASP ZAP [3], ki velja za enega izmed boljših prostodostopnih testerjev [4] za odkrivanje pogostih lukenj z metodo grobe sile. Te tehnologije so tudi široko uporabljene, iz česar sledi, da so dobro dokumentirane, kar je drugi glavni razlog za izbor. Na spletu obstaja več organizacij, ki v določenih časovnih intervalih izdajajo sezname, na katerih so navedeni najpogostejši načini zlorabe spletnih aplikacij. Ena izmed tovrstnih vodilnih organizacij je ameriški The Open Web Application Security Project (OWASP) [5]. Njihov seznam desetih najpogostejših pomanjkljivosti citira tudi sama dokumentacija PHP [6]. V tem diplomskem delu sem posebno

pozornost posvetil vdorom z uporabo injekcije, nepravilnemu upravljanju z avtentikacijo in sejami (angl. Broken Authentication and Session Management), Cross-Site Scripting-u (XSS), Cross-site request forgery (CSRF) ter izpostavitvi občutljivih podatkov. Naloga ni usmerjena le v zagotavljanje varnosti, ampak tudi v realizacijo funkcionalnosti trgovine. Razviti so trije glavni moduli spletne trgovine, glede na vloge, ki se pri poslovanju pojavljajo. To so modul za kupce (nakupovanje), modul za trgovce (CMS ali sistem za upravljanje vsebin) ter modul za dobavitelje (naročanje produktov). Razvite funkcionalnosti in njihovo delovanje so v tem delu tudi prikazane s slikami. Ker gre pri veliki množici spletnih aplikacij v končni fazi le za dostopanje do podatkovne baze in izmenjavo podatkov z njo (podobno kot pri tej nalogi) ocenjujem, da je poznavanje konceptov, ki so opisani, pomembno.

V drugem poglavju je opisan širši pregled področja, ki se ukvarja s spletnimi aplikacijami. V tretjem poglavju so na kratko predstavljene najpogostejše varnostne pomanjkljivosti spletnih aplikacij. Le-te sem želel v svoji implementaciji v čim večji meri odpraviti. V četrtem poglavju je opis implementacije, ki vključuje konkretne prikaze ranljivosti v PHP kodi. Opisano je tudi, na kakšen način sem se jim sam poskušal izogniti. V petem poglavju so predstavljeni rezultati v obliki zaslonskih slik dokončane aplikacije. Vsaka zaslonska slika prikazuje posamezno funkcionalnost spletne trgovine. Poglavje je namenjeno tudi opisu rezultatov, ki jih je ob končnem testiranju varnosti vrnil OWASP ZAP.

Poglavje 2

Pregled področja

Z vidika razvijalca se danes spletne tehnologije hitro in vsestransko razvijajo, zato je za implementacijo posamezne funkcionalnosti na voljo ogromno različnih pristopov. Imamo veliko svobode pri izbiri programskega jezika. Na spletu je precej dobro indeksirane dokumentacije o tem, kako lahko posamezen problem rešimo (npr. portal StackOverflow [7]) in najdemo lahko konkretne primere delujoče kode. Poleg tega obstaja cela vrsta internetnih skupnosti, ki so pripravljene tako ali drugače pomagati ali sodelovati. Vse našteto seveda močno pospeši proces razvoja. Vodilni jeziki [8], s katerimi se realizira zaledne dele aplikacij (angl. back-end) so PHP, C#, Java, Python, Ruby, Perl in SQL za povezovanje in interakcijo s podatkovimi bazami. Za implementacijo funkcionalnosti uporabniškega (angl. front-end) dela se najbolj uporablja JavaScript (s knjižnico jQuery). Za druge tovrstne jezike je značilno, da so odvisni od tega ali jih naš brskalnik podpira (na primer ActionScript [9]) in za delovanje ponavadi potrebujejo dodatne vtičnike.

Leta 2005 se je za skupek tehnologij, ki omogočajo izdelavo asinhronih spletnih strani, uveljavilo ime AJAX [10] (asinhroni JavaScript in XML). Z uporabo AJAX-a si lahko spletna stran brez ponovnega nalaganja izmenjuje podatke s strežnikom, kar pomeni boljšo uporabniško izkušnjo.

Današnji trendi so, da se računalniške storitve selijo v oblak. To v praksi pomeni, da podjetja kupijo ali najamejo strežnike, na katerih ponujajo svoje storitve preko spleta. Uporabniki jih lahko uporabljajo kadarkoli in kjerkoli - velikokrat je edina omejitev dostop do interneta. Take storitve so na primer DropBox [11], GoogleDrive [12] in Mozy [13].

Spletne trgovine (Amazon [14], eBay [15]) sicer niso tehnološka novost, vendar drži dejstvo, da vsako leto več ljudi tako ali drugače kupuje preko spleta [16]. Na voljo imamo ogromno transakcijskih rešitev (PayPal [17], MasterCard [18]), ki ob pravilni uporabi zagotavljajo dokaj varen način nakupovanja. Poleg tega so pomembna novost spletne denarne valute, kot je na primer BitCoin [19], s katerim se že da poslovati v nekaterih spletnih trgovinah. Razviti so tudi dobri priporočilni sistemi (Google Ads [20]), ki na podlagi iskalnih nizov profilirajo posameznika in mu predlagajo izdelke. Razvoj spletnih trgovin si lahko poenostavimo tudi z že narejenimi ogrodji za upravljanje vsebine, kot sta Wordpress [21] in Joomla [22]. Odločil sem se, da pri diplomski nalogi omenjenih ogrodij ne bom uporabljal, saj bi s tem zašel v preveč poenostavitev.

Na drugi strani se odpirajo tudi problematična vprašanja, kot sta zasebnost in varnost. Zbiranje podatkov je velikokrat nenadzorovano in uporabniki se premalo zavedajo, kolikšen del zasebnosti so pustili na spletu ob uporabljanju storitev. Sorazmerno z velikostjo računalniških sistemov raste tudi njihova kompleksnost, kar povzroči, da je težko odpravljati vse izjeme in pomanjkljivosti. Tega se dobro zavedajo tisti, ki splet zlorabljajo v kriminalne namene in to s pridom izkoriščajo. Področje prava za tehnološkimi novostmi ponavadi zaostaja. Vse več podatkov se seli s papirja na digitalne vire, zato smo postali v tem pogledu ranljivi. Če nam kdo, ki ni pooblaščen, vdre v pomembno podatkovno bazo, je to lahko razlog za propad podjetja ali ogrozitev zasebne varnosti. Na drugi strani je pozitiven vidik, da kriptografski protokoli (TLS in SSL) na zadovoljivi ravni rešujejo veliko problemov.

Poglavje 3

Predstavitev pogostih varnostnih pomanjkljivosti spletnih aplikacij

3.1 Injekcije

Injekcije so po najširši definiciji opredeljene kot napad, pri katerem napadalec skuša v obstoječo kodo vriniti dodatne ukaze, ki v aplikaciji prvotno niso napisani. S temi svojimi dodatnimi ukazi lahko manipulira s fukncionalnostmi na načine, ki za razvijalce in uporabnike niso zaželjeni. Poznamo več vrst injekcij, najpogostejša pa je SQL injekcija, pri kateri gre za spreminjanje SQL ukazov. Napadalec lahko na primer nepooblaščeno dostopa do podatkov, jih spreminja ali briše. Glavni način, kako preprečujemo injekcije, je prečiščevanje vseh spremenljivk v poizvedbi, ki so na kakršenkoli način odvisne od zunanjega uporabnika aplikacije. V PHP imamo na voljo več vgrajenih funkcij za prečiščevanje nizov (*mysql_real_escape_string()*). Če želimo še bolj zanesljivo in strukturirano prečiščevanje, lahko uporabimo še t.i. pripravljene stavke in parametrizirane poizvedbe. Pri slednjih je potrebno omeniti razred PDO (PHP Data Object), katerega sem uporabil za realizacijo poizvedb. Injekcije se na OWASP seznamu pojavijo kot najpogostejši napad, ki lahko naredi zelo veliko škode. Posledice so lahko izguba zaupanja podjetju (Sony Playstation leta 2011 [23]), kraja identitete in izguba nadzora nad sistemom.

3.2 XSS

XSS ali Cross-site scripting je napad, pri katerem napadalec aplikaciji pripiše kodo, ki se ponavadi izvede drugemu uporabniku, ko uporablja aplikacijo. Obstaja več načinov:

- Odbit (angl. Reflected) - zlonamerna koda je posredovana preko URL naslova.
- Shranjen (angl. Stored) - zlonamerna koda je shranjena v podatkovno bazo spletne strani.
- DOM osnovan (angl. DOM based) - spreminjanje kode na strani klienta (client-side) v brskalniku.

Zlonamerna koda se izvede na strani klienta. Klient (žrtev) lahko ob izvajanju te kode napadalcu razkrije kakšne varovane podatke (na primer številko seje), ali nevede obišče oziroma napade neko tretjo spletno aplikacijo. Glavni način, kako preprečimo, da bi bilo na naši spletni strani možno izvesti tak napad je, podobno kot pri injekciji, prečiščevanje vnosov uporabnikov. HTML (Hyper Text Markup Language) značke so gradniki označevalnega jezika HTML. Zavedati se moramo, da če je uporabniku strani omogočeno vsebini naše strani dodati njegove HTML značke, potem lahko znotraj njih izvaja tudi operacije z JavaScript ali z drugimi jeziki. Tega si gotovo ne želimo. V PHP imamo zato definirani funkciji *htmlspecialchars()* in *htmlentities()*, ki vse posebne HTML znake pretvorita iz njihove funkcionalne oblike (značke, narekovaji...) v izključno grafično obliko. Z drugimi besedami - odvzameta HTML funkcionalnost znakov in jih le izrišeta. Obstaja tudi vgrajena knjižnica HTMLPurifier, ki zagotavlja še bolj strukturirano očiščevanje z mnogimi dodatnimi opcijami.

3.3 CSRF

CSRF ali Cross-site request forgery je napad, pri katerem napadalec ponaredi zahtevo in jo na skrit način podtakne avtenticiranemu uporabniku, da jo le-ta izvede na strani, kjer je avtenticiran. Pri nadaljnji razlagi sta uporabljena pojma forma in zahtevek. Forma je gradnik v HTML, ki si ga lahko predstavljamo kot obrazec. Forme so lahko že izpolnjene, ali pa jih mora uporabnik izpolniti sam. Ko formo pošljemo strežniku, jo strežnik sprejme v obliki zahtevka.

Recimo, da ima prijavljeni uporabnik (žrtev) na voljo formo X, ki jo lahko izpolni in pošlje sistemu v obliki zahtevka. Neprijavljeni napadalec te forme seveda ne more izpolniti in poslati zahtevka, ker ni avtenticiran. Predpostavimo, da napadalec natančno pozna:

- obliko forme X,
- kako strežnik sprejme zahtevek forme X,
- URL naslov strani, na kateri se forma prikaže avtenticiranemu uporabniku.

Predpostaviti moramo tudi, da se oblika zahtevka, ki ga strežnik sprejme, ne spreminja dinamično. Potem lahko napadalec na neki tretji spletni strani, poimenujmo jo S, ustvari svojo formo Y, ki ima enako strukturo parametrov, kot forma X. Napadalec vključi tudi možnost, da se forma Y ob obisku spletne strani S avtomatično pošlje na naslov, kamor bi se poslal zahtevek forme X. Če vse to drži, je potrebno prijavljenega uporabnika le še prepričati, da obišče spletno stran S. Seveda obstaja še veliko drugačnih načinov, kako izvedemo CSRF. Kot vidimo, mora biti za uspešen napad ponavadi izpolnjenih veliko pogojev in napadalec mora dobro poznati sistem.

Eden izmed načinov, kako preprečiti CSRF je, da dinamično spreminjamo obliko zahtevka, ki ga strežnik sprejme. Obliko naredimo odvisno od seje prijavljenega uporabnika, zato je napadalec ne more uganiti na preprost način. CSRF v povezavi z XSS pa je zelo nevarna kombinacija, ker napadalcu omogoča obiti skoraj vse obrambne mehanizme proti CSRF [24].

3.4 Neustrezno hranjenje podatkov

V podatkovni bazi med drugim hranimo občutljive podatke, kot je na primer geslo. Za gesla je priporočljivo, da jih pred shranjevanjem v podatkovno bazo zgostimo s pomočjo zgoščevalne funcije (angl. hash function). Kot primer lahko navedemo zgoščevalni funkciji MD5 in SHA1. Ko se uporabnik želi prijaviti v sistem, vpiše geslo. Njegov vnos zgostimo z enako zgoščevalno funkcijo, kot je zgoščen zapis gesla (izvleček) v bazi. Če se niza ujemata, je uporabnik vnesel pravilno geslo. To delamo za primer situacije, v kateri bi imela nepooblaščena oseba priložnost za kratek čas videti vsebino podatkovne baze. Prva lastnost zgoščevalnih funkcij, ki nam oteži razbitje izvlečka je, da ne glede na dolžino vhodnega niza, vedno vrnejo enako dolg izhodni niz. Naslednja koristna lastnost zgoščevalnih funkcij je, da ob majhni spremembi vhodnega niza dobimo močno spremenjen izhodni niz. Zato za razbitje potrebujemo postopek z grobo silo (angl. brute-force), ki je natančneje opisan v poglavju 4.6.3 Prijava v sistem. Ta postopek je časovno precej potraten, še posebej če ne vemo, kolikšna je dolžina vhodnega niza. V implementaciji te naloge je uporabljena zgoščevalna funkcija SHA1.

3.5 Nepravilno upravljanje z avtentikacijo in sejami

Pogosti razlogi za uspešen vdor so:

- Dopuščanje nastavitve predvidljivih uporabniških imen in gesel.
- Nekriptirane povezave.
- Nepravilno upravljanje s spremenljivkami seje.

Sistem, ki skrbi za registracijo novih uporabnikov, mora zagotavljati, da si uporabniki ne morejo izbirati predvidljivih uporabniških imen in gesel (na primer "1234567"in "password"). Take podatke lahko z lahkoto uganemo. MITM (Man-in-the-middle) [25] je napad, pri katerem se napadalec postavi med pošiljatelja in prejemnika. Ko sporočilo potuje, ga le-ta prestreže in poskuša dešifrirati. Če pošiljanje prijavnega obrazca ni kriptirano, je možno z napadom MITM prestreči

zahtevek in iz njega enostavno prebrati podatke za avtentikacijo. Povezave moramo zato kriptirati. Ko se uspešno prijavimo v sistem z uporabniškim imenom in geslom, nam je v spletnih aplikacijah ponavadi dodeljena unikatna identifikacijska številka (ID) seje. Slednja omogoča, da nam ni potrebno ob vsakem nadaljnjem zahtevku, ki ga po prijavi pošljemo strežniku, ponovno pisati uporabniškega imena in gesla. Preveri se le, ali je ID seje nastavljen. HTTP (Hypertext Transfer Protocol) je nabor pravil, ki govorijo o tem, kako morajo biti sporočila, ki se prenašajo preko svetovnega spleta, sestavljena in poslana. HTTP ne pomni prejšnjih akcij na strežniku. To ponovno odpre možnost napada, ki mu pravimo ugrabitev seje (angl. session hijacking). Splošna ideja je v tem, da napadalec žrtvi ukrade ID seje in se z njim prijavi v sistem, pri čemer dobi vse pravice žrtve. Nekaj let nazaj smo lahko brali o Firefox vtičniku FireSheep [26], s katerim se je bilo mogoče zgolj z ukradenim piškotkom (angl. cookie), v katerem je bila shranjena številka seje, prijaviti v tuje profile uporabnikov znanih socialnih omrežij. To je predstavljalo veliko in nevarno varnostno luknjo. Prva stvar, s katero povečamo varnost je, da namesto hranjenja ID seje v piškotku, za hranjenje uporabimo spremenljivko seje (angl. session variable). URL (Uniform Resource Locator) je enolični naslov spletne strani, ki mu po slovensko rečemo "enolični krajevnik vira". Prav tako ID seje ne smemo izpostavljati v URL naslovu. Zagotovljeno mora biti, da se ta spremenljivka generira na čim bolj nepredvidljiv način, pri procesu odjave pa se mora uničiti. Od tod sledi vprašanje, na kakšen način PHP generira spremenljivko seje. Ker si lahko pogledamo izvorno kodo tega jezika [27], ugotovimo, da se spremenljivka generira s pomočjo zgoščevalne funkcije MD5, vhodni parametri pa so:

- IP naslov obiskovalca.
- Trenutni čas.
- PHP psevdo naključni generator.
- Naključni vir operacijskega sistema (angl. OS random source).

Vidimo, da zaradi lastnosti vhodnih parametrov ne moremo zagotoviti popolne unikatnosti. Je pa verjetnost, da bi dvakrat dobili enak ID dovolj majhna, kar povzroči, da je varnost zagotovljena v zadostni meri. Priporočljivo je tudi, da ID seje preteče, če uporabnik določen čas ni aktiven.

3.6 Neustrezna konfiguracija strežnika

Ko stran po razvoju objavimo na spletu, moramo izključiti vsa opozorila strežnika, ki opozarjajo na napake, ki se pojavljajo na strani strežnika (angl. server-side). Če napadalec diagnozo napake vidi, jo lahko uporabi za napad. Poleg tega je priporočljivo, da izključimo možnost prikazovanja vsebine v mapi oziroma direktoriju (angl. directory listing). Predstavljajmo si, da strežniku podamo URL naslov, ki definira le mapo, ne pa datoteke v njej. Poleg tega mapa ne vsebuje datoteke "index.html", na katero bi bili privzeto preusmerjeni. Če ima strežnik vključeno omenjeno opcijo, bo obiskovalcu te strani prikazal vse datoteke, ki se nahajajo v podani mapi. Ta podatek je lahko osnova za napad.

3.7 CAPTCHA

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) je test, katerega cilj je ugotoviti, ali je v interakciji z našo aplikacijo človek ali računalnik. Računalniški agent (angl. internet bot) je program, ki avtomatično izvaja akcije preko spleta. Agenta med drugim lahko programiramo tako, da z visoko frekvenco samodejno izpolnjuje obrazce, ki se pojavljajo na določenih spletnih straneh. V primeru naše spletne trgovine bi tak agent lahko povzročil, da bi imeli v nekaj sekundah v naši podatkovni bazi na tisoče novo prijavljenih uporabnikov, ki bi bili fiktivne osebe. Tega si seveda ne želimo. V preteklosti je bilo veliko poskusov, kako tovrstne težave odpraviti. Vse rešitve temeljijo na zadajanju neke naloge, ki je za človeka relativno enostavna, za umetno inteligenco pa mora biti čim bolj časovno zahtevna. Naloga je močno odvisna od tega, koliko je umetna inteligenca napredna. Za prve CAPTCHA sisteme je tako zadoščalo, da je moral uporabnik prepisati niz popačenih znakov, ki so bili predstavljeni v grafični obliki. Lahko je bil zahtevan tudi krajši izračun in podobno. Za računalniški vid danes razpoznavanje črk in številk s slike ne predstavlja več velike težave, zato je bilo potrebno razviti boljše načine. V času pisanja te diplomske naloge je Google razvil tako imenovan "No CAPTCHA reCAPTCHA". Sistem omogoča boljšo uporabniško izkušnjo, saj je uporabniku treba le klikniti na določeno točko. Preverja se premik miške, časovna odzivnost in par privatnega

ter javnega ključa. Če je sistem po dobljenih podatkih še vedno v dvomu, moramo izpolniti še uganko, ki temelji na razpoznavanju semantične vsebine s slik. Reševanje tovrstnih problemov je za računalnik še vedno zelo kompleksno.

3.8 Phishing

Phishing oziroma spletno ribarjenje je napad, pri katerem napadalec izkoristi zaupanje žrtve, ta pa mu sama izda občutljive podatke. Poznamo več variacij tega napada. Izveden je lahko tako, da je žrtvi poslana elektronska pošta, v kateri se napadalec pretvarja, da je zaupanja vredna organizacija in od nje zahteva podatke (uporabniško ime in geslo). Drugi način je manipulacija s povezavami, pri kateri napadalci postavijo spletno stran, ki ima identičen izgled kot prava, a so v URL naslovu nekoliko spremenjene ali zamenjane črke. Tako je prevaro težko zaznati. Tovrstne napade v sistemu lahko zaznamo z beleženjem vseh naprav, s katerih se je uporabnik prijavljal vanj. Uporabnika se nato obvešča, da so bile prijavljene nove naprave. Po možnosti se teh obvestil ne da dokončno zbrisati, saj bi jih v nasprotnem napadalec lahko izbrisal sam in s tem zakril svojo sled. Drugi pristop je, da prijavo dodatno zavarujemo s certifikati. V moji spletni trgovini so uporabljeni certifikati X.509.

Poglavje 4

Implementacija

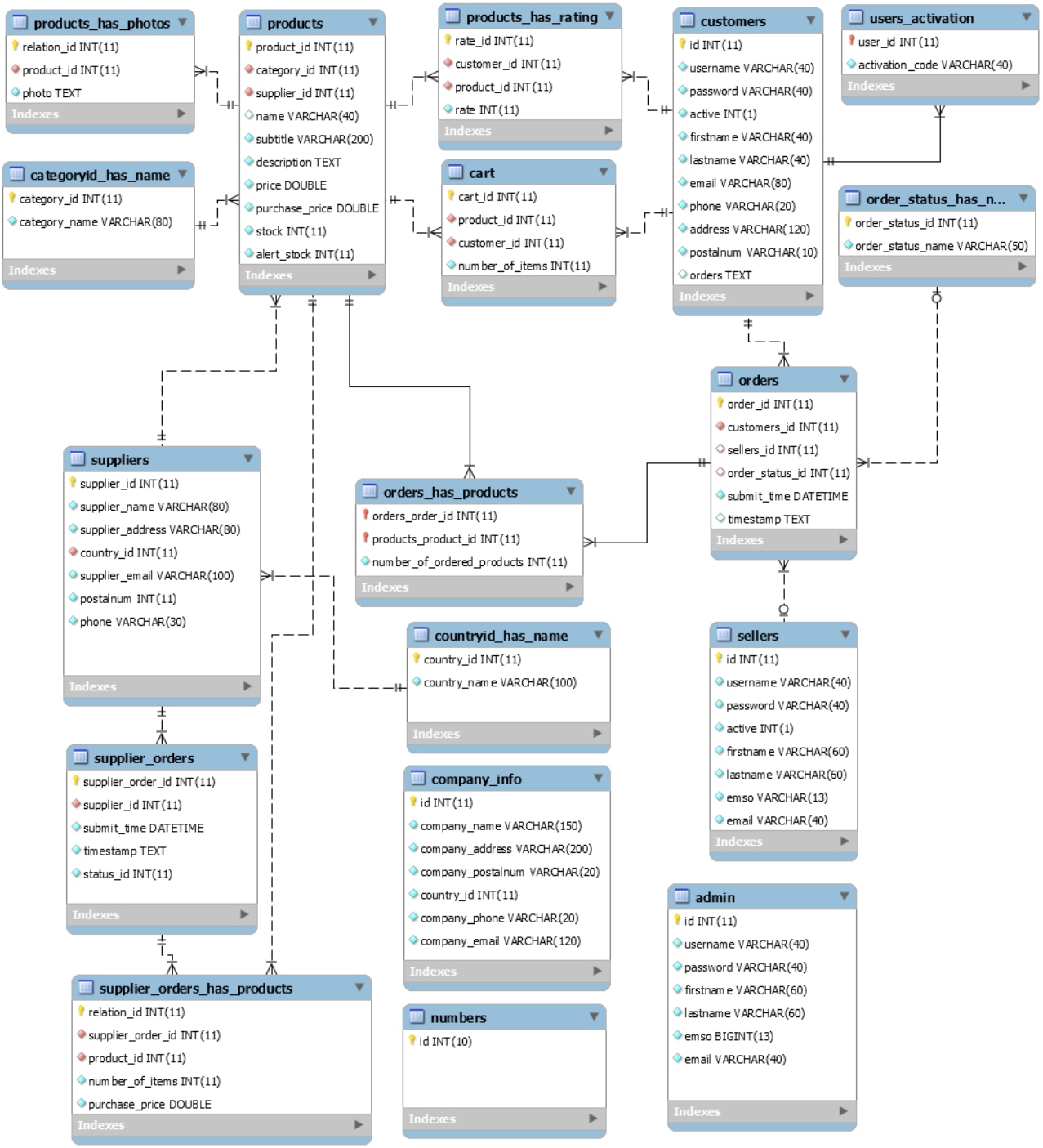
4.1 Načrtovanje podatkovne baze

Podatkovno bazo moramo na začetku dobro načrtovati, saj so kasnejši popravki njene strukture težji. Popravljati moramo namreč vse poizvedbe, ki smo jih napisali v programskem jeziku, ki opisuje logiko spletne strani. Pri načrtovanju je priporočljivo, da se držimo načel (vsaj) tretje normalne oblike. Če na hitro povzamemo teorijo normalizacije podatkovnih baz [28], pomeni tretja normalna oblika, da mora biti podatkovna baza najprej v prvi in drugi normalni obliki.

- Prva normalna oblika določa, da sme domena posameznega atributa vsebovati le atomarne (nedeljive) vrednosti, kar pomeni, da v posameznem polju ne smemo imeti seznamov.
- Druga normalna oblika določa, da morajo biti vsi atributi, ki niso del ključa, odvisni od celotnega ključa. Vse atribute, za katere to ne velja, moramo prestaviti v novo relacijo.
- Tretja normalna oblika pravi, da morajo biti vsi atributi, ki niso del ključa, odvisni samo od ključa (ne od kakšnih drugih atributov, ki niso del ključa). Temu pravimo tudi, da ni tranzitivnih odvisnosti.

Naša struktura podatkovne baze je prikazana na sliki 4.1, izdelana pa je v okolju MySQL Workbench. V podatkovni bazi so semantično najbolj pomembne tabele products, customers, sellers in suppliers. Tako, kot je mogoče razbrati iz imen,

gre za vloge, ki nastopajo v spletni trgovini. Poslovanje poteka preko naročil, kar predstavljata tabeli orders (kupci naročajo prek naše spletne trgovine) in supplier_orders (spletna trgovina naroča pri dobaviteljih). Kupci lahko ocenjujejo produkte (tabela products_has_rating) ali jih dajejo v svoj voziček (angl. cart). Vsak produkt ima lahko več slik (products_has_photos) in je dodeljen v neko kategorijo (categoryid_has_name). Ker imamo v nekem naročilu lahko več produktov in en produkt v večih naročilih, to relacijo M:N rešujemo z vmesno tabelo orders_has_products in podobno supplier_order_has_products. V tabeli company_info imamo podatke o naši spletni trgovini.



Slika 4.1: Struktura podatkovne baze.

4.2 Programska struktura projekta

Konceptualno je programska koda projekta razdeljena v mape, ki se nahajajo v korenski mapi, glede na podsisteme in module, ki se pojavljajo. Te mape so:

- **database** - povezovanje s podatkovno bazo in funkcije, ki izvajajo vnaprej pripravljene poizvedbe.
- **constants** - deli kode, ki so značilni za več spletnih strani hkrati, namenjeni večkratnemu vključevanju.
- **admin** - skripta logike, do katere lahko dostopa le s certifikatom prijavljen administrator.
- **sellers** - skripta logike, do katere lahko dostopajo le s certifikatom prijavljeni prodajalci.
- **customers** - skripta logike kupcev.
- **products** - skripta za upravljanje s produkti.
- **img** - slike produktov.
- **lib** - knjižnice, ki sem jih sam spisal za potrebe tega projekta.
- **js** - JavaScript knjižnice in koda, ki je vključena na druge strani.
- **css** - oblikovna plat spletne strani.

Definiran je tudi "api.php", ki služi kot REST API [29] (Representational State Transfer Application Programming Interface). To pomeni, da je realiziran aplikacijski vmesnik, ki se ga da na preprost način uporabiti v primeru, ko bi druge aplikacije želele dostopati do javno vidnih podatkov o produktih, ki so objavljeni v naši trgovini.

4.3 Upoštevanje načel varnega programiranja s konkretnimi primeri

4.3.1 Preprečevanje zlonamerne kode

Obrambni mehanizem pred SQL injekcijo je v našem programu prikazan na sliki 4.2. Vse spremenljivke, ki jih je vnesel uporabnik (v primeru so vnešene preko metode GET), so najprej prečiščene. Napisal sem knjižnico "validation_functions.php", ki vsebuje validacijske funkcije za vsako vrsto vhodov posebej (ID, elektronska pošta, naslov, poštna številka...). Funkcije vrnejo vhodni niz, če je ustrezen, sicer vrnejo NULL. Če stran na primer od uporabnika zahteva elektronsko pošto, se iz knjižnice kliče funkcija *validateEmail()*, ki niz testira z regularnim izrazom. PHP dokumentacija namreč priporoča uporabljati t.i. način preverjanja z "belimi seznami"(angl. whitelisting) namesto načina s "črnimi seznami"(angl. blacklisting). Whitelisting je definiranje dopustnih možnosti, blacklisting je ravno nasprotno navajanje prepovedanih. Razlog, zakaj je blacklisting slabša izbira, je velika verjetnost, da bomo ob naštevanju prepovedi kakšno izpustili, kar predstavlja potencialno možnost vdora. Če imamo neko SQL poizvedbo, katere del je uporabnikov vhod, se pred injekcijo dodatno zavarujemo še s pripravljenimi stavki in parametriziranimi poizvedbami knjižnice PDO, kot je prikazano na sliki 4.2.

```php
<?php
if(isset($_GET['email'])) {
        $email = validateEmail($_GET['email']);
        if( !is_null($email) ) { //ustrezen vhod

                $db = DBInit::getInstance(); //PDO
                $sql = "UPDATE ... :email ...;"; //hipoteticna poizvedba
                $query = $db->prepare($sql);
                $query->bindValues(":email", $email);
                $query->execute();

                //DO SOMETHING

        } else { //neustrezen vhod
                //DO SOMETHING ELSE
        }
}
?>
```

Slika 4.2: Način obrambe pred SQL injekcijo.

Kot sem že omenil, se XSS napada lahko ubranimo s prekodiranjem HTML značk. Zelo pomembno je, da kakšnega vnosnega polja ne pozabimo prečistiti. Primer je predstavljen na sliki 4.3. Če si predstavljamo, da funkcije *htmlspecialchars()* ne bi bilo in bi preko GET zahtevka podali vrednost x = $'$ >< *script* > *alert*(); < /*script* >$'$, bi nam preko URL naslova uspelo zagnati JavaScript kodo. Pri tem je sicer olajšujoče dejstvo, da ima večina novejših različic razširjenih spletnih brskalnikov vključeno avtomatsko čiščenje URL naslovov pred procesiranjem. Preprost način t.i. odbitega XSS napada, ki je omenjen v primeru, torej ne bi vplival na uporabnike posodobljenih brskalnikov. Za testiranje varnosti lahko to možnost seveda tudi izključimo (v Google Chrome z argumentom –disable-xss-auditor [30]). Še vedno pa obstaja možnost, da bi na podoben način napadalcu uspelo naložiti JavaScript kodo v podatkovno bazo (shranjen XSS) spletne strani in bi jo logika strani ob poizvedbi tako sama vključila v vsebino. V takem primeru nam lahko pomagajo vtičniki (na primer NoScript za Firefox), ki jih veliko uporabnikov spleta ne uporablja.

```
<?php
if(isset($_GET['x'])) {
        $x = htmlspecialchars($_GET['x']);
}

<div id="<?= $x ?>">
        <p>Text.</p>
</div>
}

?>
```

Slika 4.3: Preprost primer zaščite pred XSS napadom.

Pred CSRF se lahko zavarujemo tako, da dinamično spreminjamo imena parametrov forme. Koda, ki prikazuje CSRF ranjlivost, je prikazana na sliki 4.4, nadaljevanje je njen opis. Definirajmo tri spletne strani: "varovana.php", "druga.html" in "nevarnost.html". Naj bo "varovana.php" stran, na kateri moramo biti prijavljeni, če želimo pisati v datoteko "data.txt". Prijavimo se preprosto tako, da pritisnemo na gumb "Get session" (gre za poenostavitev - na pravi spletni strani bi morali vpisati uporabniško ime in geslo). Strani "druga.html" in "nevarnost.html" je modificiral napadalec. Naj bo na "druga.html" vključena samo skoraj nevidna "iframe" HTML značka, ki omogoča vključevanje vsebine "nevarnost.html" na "druga.html". V "nevarnost.html" imamo izpolnjeno formo, katere parametri imajo enako ime, kot v formi znotraj "varovana.php". Vsebuje tudi JavaScript kodo, ki poskrbi, da se forma ob obisku te spletne strani (preko metode POST) samodejno pošlje logiki strani "varovana.php". Če bo na "varovana.php" prijavljeni uporabnik obiskal "druga.html", bo (nevede) v datoteko "data.txt" zapisal niz, ki je definiran v parametru "nevarnost.html". Naloga napadalca je torej, da prijavljenega uporabnika zvabi na "druga.html".

Zelo pogosta implementacija obrambe je t.i. sinhronizator žetonov (angl. synchronizer token design pattern) [31]. Ideja je sledeča: ko se uporabnik prijavi, generiramo unikaten žeton, ki ga shranimo v sejno spremenljivko. Vključimo ga tudi kot del vsake forme. Tako se ob pošiljanju zahtevka te forme žeton pošlje kot del zahtevka. Strežnik procesira zahtevek le, če se žetona v sejni spremenljivki in v poslanem zahtevku ujemata. Na tak način onemogočimo napadalcu, da bi v svojo ponarejeno formo vključil žeton. V naši spletni trgovini je obramba pred CSRF realizirana pri pošiljanju naročila. Če bi bilo omogočeno kupovanje s kreditno

kartico, bi bil to eden izmed bolj občutljivih delov aplikacije, saj gre pri njem za transakcijo.

```
############ varovana.php ############
<?php
session_start();
if(isset($_POST['getSessionID'])) $_SESSION['id'] = rand();  //prijava
if( isset($_SESSION['id']) && isset($_POST['add'])
                           && isset($_POST['send'])  ) {
        $string = $_POST['add']."\n";
        file_put_contents("data.txt", $string, FILE_APPEND | LOCK_EX);
}
?>
<?php if(!isset($_SESSION['id'])): ?> //nismo prijavljeni
        <form method="POST" action="">
                <input type="submit" name="getSessionID"
                                     value="Get session">
        </form>
<?php else: ?> //smo prijavljeni
        <form method="POST" action="">
                <input type="text" name="add">
                <input type="submit" name="send">
        </form>
        <?= $output = file_get_contents("data.txt"); ?>
<?php endif; ?>
####################################
############ druga.html ############
<html>
<body>
<iframe src="nevarnost.html" width="1" height="1"></iframe>
</body>
</html>
########################################
############ nevarnost.html ############
<html>
        <body onload="setTimeout('document.f.submit();', 1)">
        <form name="f" action="varovana.php" method="POST">
                <input type="hidden" name="add" value="insider">
                <input type="hidden" name="send">
        </form>
</body>
</html>
```

Slika 4.4: Prikaz CSRF ranljivosti.

4.3.2 Izdelava certifikatov in konfiguracija strežnika Apache

Izdelave samopodpisanih certifikatov sem se lotili s programom TinyCA2 [32], v katerem sem izdelal svojo certifikatno agencijo in certifikate za administratorja ter prodajalce. Po zgledu mnogih spletnih trgovin, certifikatov za kupce nisem izdeloval. To pomeni, da je nakupovanje sicer res manj varno, a sem s tem izboljšal uporabniško izkušnjo. Postopek izdaje in priprave certifikatov je namreč za uporabnike preveč dolgotrajen in kompleksen, kar stranke odbija.

Na strežniku Apache (uporabljal sem lokalni strežnik WAMP) se za vzpostavitev SSL povezavave v konfiguracijski datoteki "httpd.conf"vključi modul SSL ter datoteko "httpd-ssl.conf". Nato se v datoteki "php.ini"vključi razširitev (angl. extension) "php_openssl.dll". V datoteki "httpd-ssl.conf"nato nastavimo vse glavne parametre [33], ki jih potrebujemo za vzpostavitev SSL. To so:

- SSLEngine (vklop modula).
- SSLCertificateFile (kazalec na datoteko, v kateri imamo samopodpisano strežniško digitalno potrdilo).
- SSLCertificateKeyFile (kazalec na datoteko, v kateri se nahaja zasebni ključ).
- SSLCACertificateFile (kazalec na datoteko, v kateri je digitalno potrdilo certifikatne agencije).
- SSLCARevocationFile (kazalec na seznam preklicanih digitalnih potrdil).

Poleg tega lahko za vsako hierarhično stopnjo datotek naše spletne strani v značkah Directory definiramo dodatne SSL možnosti. Primer splošnih pravil in pravil mape, v katero je možno vstopiti le s certifikatom administatorja, je prikazan na sliki 4.5.

```
<VirtualHost>
        ...
        <Directory spletnaTrgovina/admin/>
                AllowOverride None
                SSLVerifyDepth 1
                SSLVerifyClient require
                SSLOptions +ExportCertData +StdEnvVars
                SSLRequire %{SSL_CLIENT_S_DN_CN} eq "Admin"
        </Directory>

        SSLEngine on
        SSLCertificate ".../cert.pem"
        SSLCertificateKeyFile ".../key.pem"
        SSLCACertificateFile ".../CAcert.pem"
        SSLCARevocationFile ".../CAcrl.pem"

         ...
</VirtualHost>
```

Slika 4.5: Primer nastavitve SSL pravil na strežniku Apache.

Projekt je zasnovan tako, da posamezna mapa predstavlja vlogo v sistemu. Administrator ima v projektu svojo mapo, kupci svojo in tako naprej. Iz podanega primera lahko vidimo, da je vstop v mapo (obisk strani) prepovedan, če strežniku ne pošljemo ustreznega certifikata. Stvar s tem še ni končana, saj moramo vse HTTP zahtevke na vsebino te mape preusmeriti na HTTPS. To je realizirano s pomočjo datoteke ".htaccess", ki jo vstavimo v mapo posamezne vloge (posebej v mapi administratorja, posebej v mapi prodajalca). Vsebino te datoteke najdemo na sliki 4.6.

Ob vsem definiranem ostaja še en problem. Če smo eden izmed prodajalcev, se trenutno lahko v sistem prijavimo s svojim uporabniškim imenom in geslom ter s katerimkoli certifikatom prodajalca, tudi če je certifikat tuj. Želimo si torej povezati posamezen ID prodajalca s posameznim certifikatom zato, da se ne bo mogoče prijavljati s tujimi certifikati. To sem naredil tako, da se s PHP kodo ob vsaki potrebni predložitvi certifikata testira, ali se polje za elektronsko pošto v certifikatu ujema s poljem za elektronsko pošto v podatkovni bazi (pri določenem ID-ju prodajalca). Le v primeru, da pride do ujemanja, je prijava možna.

```
<IfModule mod_rewrite.c>
        RewriteEngine on
        RewriteCond %{HTTPS}% !on
        RewriteRule (.*) https://%{HTTP_HOST}%{REQUEST_URI}
</IfModule>
```

Slika 4.6: Preusmeritev na HTTPS (.htaccess).

Naslednji ukrep je izključitev navajanja vseh strani v mapi. Na sliki 4.7 je prikazano, kaj uporabnik vidi, če obišče mapo "lib". Ker v mapi ni datoteke "index.html"vidimo celotno vsebino mape, česar si ne želimo.

Slika 4.7: Prikaz, kako strežnik navaja vse datoteke v mapi.

To lahko popravimo tako, da v datoteko ".htaccess"dodamo ukaz "Options -Indexes", ali pa to naredimo globalno v datoteki "httpd.conf" strežnika Apache. Rezultat tega popravka je, da je obisk zavrnjen.

Onemogočiti moramo tudi PHP modul za izpis napak. To naredimo tako, da v "php.ini"datoteki definiramo display_errors(false) in zapis napak preusmerimo v datoteko z log_errors(true).

4.4 Nastavitev HTTP odzivov

OWASP priporoča naslednje ukrepe:

- Onemogoči X-Frame-Option, če ne uporabljaš okvirov (prevencija pred XSS).
- Vključi opcijo HttpOnly, da koda na strani klienta ne more dostopati do zaščitenih piškotkov (prevencija pred XSS).
- Onemogoči odbit XSS z vključitvijo X-XSS-Protection.
- Če je le mogoče vključi opcijo cookie_secure, ki pravi, da morajo biti piškotki poslani preko HTTPS.
- Ustrezno nastavi Cache-control in Pragma parametra.
- Nastavi X-Content-Type-Options na *nosniff*, s čimer preprečiš MIME sniffing.

Proti XSS sem se zaščitil tako, da sem nastavil X-Frame-Options Header na *DENY*, kar pomeni, da se na stran ne morejo vključevati okviri (HTML značke "iframe"). PHP koda je prikazana na sliki 4.8. Če vseeno želimo vključevati okvire iz svoje domene, potem lahko nastavimo opcijo na *SAMEORIGIN*, vendar v našem projektu podobnega vključevanja ni. Zastavica HttpOnly pove, ali ima skripta, ki se izvaja na strani klienta, dovoljen dostop do vsebine piškotka.

```
<?php
header("Cache-Control: no-store, no-cache, must-revalidate, max-age=0");
header("Pragma: no-cache");

header("X-XSS-Protection: 1");
header('X-Content-Type-Options: nosniff');

header( 'X-Frame-Options: DENY' );
ini_set('session.cookie_httponly', 1);
ini_set('session.cookie_secure', 1);
?>
```

Slika 4.8: Implementacija OWASP priporočil.

V primeru XSS napada, bi nam lahko preko JavaScript kode ukradli avtentikacijski piškotek. Zato si ne želimo, da bi skripta lahko dostopala do njegove vsebine. Opcija *nosniff* pri X-Content-Type-Options pomeni, da je brskalniku onemogočeno MIME (Multipurpose Internet Mail Extensions) pregledovanje. Gre za ugotavljanje tipa vsebine podatkov, če tip ni definiran. Cache-control in Pragma skrbita za začasno shranjevanje vsebin, ki smo jih že pogledali, s čimer se izognemo večkratnemu nalaganju. Takšno shranjevanje OWASP odsvetuje.

4.5 Izdelava modula za administratorja in naročanje pri dobaviteljih

4.5.1 Upravljanje produktov, prodajalcev in dobaviteljev

Administrator ima v trgovini najvišjo hierarhično vlogo z največ pravicami. Prijaviti se mora z unikatnim certifikatom. Njegove funkcije so, da lahko v bazo dodaja (ali spreminja) prodajalce, produkte, kategorije produktov in dobavitelje. Lahko spreminja tudi podatke o podjetju, ki vodi spletno stran, in seveda podatke svojega profila. Administrator ima v naši spletni trgovini tudi nalogo, da od dobaviteljev naroča nove produkte. Ker je, za razliko od prodajalcev, ki opravljajo bolj operativne naloge, njegova vloga strateška, nima nekaterih funkcionalnosti prodajalcev (odobritev/zavrnitev naročil kupcev).

Če si bolj podrobno pogledamo upravljanje s prodajalci, ima administrator možnost vključitve in izključitve posameznega prodajalca. To služi v primeru, ko nek prodajalec začasno ne dela (dopust, porodniška in podobno). Takrat le enostavno izključimo možnost njegove prijave (dodatna varnost) in ni potrebno brisati celotnega profila prodajalca. Na tem mestu je pomembno še, da se elektronska pošta prodajalca, ki ga upravljamo tu, ujema z elektronsko pošto, ki je zabeležena na njegovem certifikatu.

Pri funkcionalnosti upravljanja s produkti moramo vsakemu produktu (poleg osnovnih podatkov) navesti tudi nabavno ceno in prodajno ceno. Tako lahko na koncu meseca na preprost način izvajamo različne izračune, hkrati pa imamo olajšano generiranje predračunov pri naročanju od dobaviteljev. Naslednja zahtevana parametera sta zaloga in meja zaloge, kjer nas program sam opozori, naj naročimo novo količino tega produkta, kadar se zaloga spusti pod mejo. Seveda lahko tega priporočila pri naročanju (ki ga bom opisal pozneje) tudi ne upoštevamo. Poglaviten namen meje zaloge je narediti ob naročanju neko učinkovito in preprosto razmejitev med tistimi produkti, ki jih je že potrebno na novo naročiti in tistimi, ki jih imamo še dovolj na zalogi. Za vsak produkt je možno dodati tudi več slik, zato se - zaradi ohranjanja strukturiranosti na strežniku - za vsak produkt posebej ustvari mapa, kamor se le-te shranijo. Funkcionalnost dodajanja in spreminjanja kategorij je namenjena temu, da bi bila koda trgovine čim bolj široko uporabna ter bi jo lahko uporabljalo več tržnih niš. Pri funkcionalnosti upravljanja z dobavitelji

moramo biti pozorni na to, da se bodo avtomatsko generirane naročilnice pošiljale na elektronsko pošto, ki jo bomo podali v formi znotraj nje.

4.5.2 Izdelava naročilnic in sistem, ki pomaga pri odločanju

Funkcionalnost naročanja produktov je narejena tako, da si moramo od vseh dobaviteljev najprej izbrati tistega, ki mu želimo oddati naročilnico. Takoj po izboru se nam prikažejo vsi produkti, ki nam jih ta dobavitelj dobavlja. Razdeljeni so na tiste, katerih zaloga je že padla pod mejo in tiste, ki se jim to še ni zgodilo. Za vsak produkt posebej si lahko pogledamo graf prodaje, ki opisuje kdaj in koliko smo jih prodali. Ta graf si lahko izrisujemo za različna časovna obdobja: zadnji teden, zadnji mesec in zadnje leto.

Izris grafa je realiziran s knjižnico D3.js [34] (Data-Driven Documents). V ta namen smo ustvarili dve PHP datoteki. Prva na podlagi parametrov (ID produkta in časa) izvede poizvedbo, koliko dotičnih produktov smo prodali na določen dan in to zakodira v podatkovni objekt JSON (JavaScript Object Notation), ki je tudi izhod te datoteke. V drugi datoteki te podatke preberemo s pomočjo omenjene knjižnice in jih izrišemo. Pri izrisu grafa je nastal problem, ker lahko iz baze dobimo le podatek, na katere datume je bilo prodanih več kot 0 produktov, ostalih točk na grafu, ki bi prikazovale, kdaj nismo prodali nobenega artikla, pa nimamo. Stvar je teoretično mogoče rešiti na nivoju JavaScript kode, vendar sem se odločil, da problem rešim na nivoju SQL poizvedbe. Praznim vrednostim je bilo torej potrebno prirediti datume, kar sem naredil s programerskim trikom. Kreiral sem novo preprosto tabelo numbers, ki vsebuje le zaporedne številke. Ker je ta SQL poizvedba zelo zanimiva, je prikazana na sliki 4.9. Notranja poizvedba z imenom T nam vrne združene vse datume od začetnega naprej, T1 vrne vse atribute o prodaji določenega produkta, ki je bil prodan, zunanja poizvedba pa ti dve tabeli združi (LEFT JOIN) glede na datum. Funkcija *Coalesce()* nadomesti vse NULL vrednosti z ničlo, kar je pravzaprav operacija, zaradi katere je bilo potrebno izvesti to poizvedbo. Na koncu se še sešteje vse produkte, ki so bili prodani na isti datum.

```
SELECT date, products_product_id AS product_id,
           SUM(COALESCE(number_of_ordered_products, 0)) AS num_ordered
           FROM (SELECT DATE_ADD(:startDate, INTERVAL n.id - 1 DAY) AS date
                FROM shop.numbers n) AS T
           LEFT JOIN
                (SELECT * FROM orders o JOIN orders_has_products ohp
                ON o.order_id = ohp.orders_order_id
                 WHERE products_product_id = :pid AND order_status_id = 1)
                 AS T1
           ON DATE(T1.submit_time) = T.date
           WHERE date<= :endDate GROUP BY date ORDER BY date;";

//  Ko velja, da je 'order_status_id' enak 1, smo produkt res prodali.
//  Spremenljivka ':startDate' predstavlja datum (teden/mesec/leto nazaj)
//  Spremenljivka ':endDate' je danes.
```

Slika 4.9: SQL poizvedba, ki jo uporabimo za izris grafa.

4.6 Izdelava modula za kupce

4.6.1 Iskalnik in kategorije

Kupce moramo najprej razdeliti v dve skupini: neprijavljene in prijavljene. Neprijavljenim so omogočene le osnovne operacije, kot je na primer iskanje in ogled podrobnosti o izdelkih. Iskalne možnosti so tri - možno pa je iskati tudi po dveh ali vseh treh hkrati. To so: iskanje po kategoriji, iskanje po iskalnem nizu in naraščajoče ter padajoče urejanje izdelkov, glede na določen kriterij. Na stran se nam privzeto razporedi devet produktov, realizirano je tudi ostranjevanje. Tukaj so uporabniku vidni samo nekateri podatki. Ko kupec klikne na določen izdelek, se prikažejo njegove podrobnosti.

4.6.2 Pregled podrobnosti izdelka

Na strani podrobnosti se prikažejo še drugi, za kupca relevantni podatki o izdelku, kot so opis izdelka, koliko izdelkov je še na zalogi in ocena izdelka, ki so mu jo dodelili predhodni kupci izdelka. Največ dela na tej strani je bilo z implementacijo

pregledovalnika slik izdelka. Pomagal sem si s prostodostopno knjižnico EasyRotator [35], ki deluje s pomočjo jQuery-ja. Lahko izbiramo pomanjšanje slike v spodnjem predelu galerije, ob kliku se nato izbrana slika poveča.

4.6.3 Prijava v sistem

Če se želimo v sistem prijaviti, moramo najprej ustvariti svoj račun tako, da se registriramo - preusmerjeni smo na povezavo HTTPS. Če ob registraciji pri vnašanju podatkov naredimo kakšne napake (vnešeni podatki ne ustrezajo regularnemu izrazu), nas sistem opozarja, dokler vseh napak ne odpravimo. V formi so zahtevani tudi podatki o uporabnikovem prebivališču, saj naša spletna trgovina prodane artikle pošilja po pošti. Ko uporabnik nastavlja svoje geslo, mora le-ta iz varnostnih razlogov ustrezati naslednjim pogojem [36]:

- Geslo mora biti dolgo od 8 do 20 znakov.
- Vsebovati mora vsaj eno malo črko in vsaj eno veliko črko.
- Vsebovati mora vsaj eno številko.
- Vsebovati mora vsaj en poseben znak, ki ni ne črka in ne številka.

To je pomembno zato, ker napadalci s preprosto tehniko grobe sile lahko ugibajo vsa možna gesla. Z zadostno dolžino gesla napadalcem močno povečamo čas ugibanja, saj število kombinacij z vsakim dodatnim znakom eksponentno naraste. Če z N označimo število različnih kombinacij, s P število vseh možnih uporabljenih znakov in z L dolžino gesla, potem velja formula $N = P^L$. V operacijskem sistemu Kali Linux [37], ki je namenjen prav testiranju varnosti sistemov, lahko najdemo veliko uporabnih orodij, ki delujejo na podlagi metode z grobo silo (John the Ripper, Aircrack-ng, Hydra, Cain & Abel). Naslednji zelo pogost napad je tako imenovan napad s slovarjem. Podobno kot pri grobi sili gre za preverjanje različnih možnosti, le da je množica poskusov veliko bolj omejena. Slovar se sestavi tako, da vanj vnašamo le nize črk s semantičnim pomenom (za razliko od napada z grobo silo, pri katerem večina nizov nima pomena). Potem množico lahko razširimo tako, da posamezne nize s semantičnim pomenom združujemo v nove nize. Za slovarje je tudi značilno, da jih morajo napadalci prirediti glede na državljanstvo in jezike, ki jih govori žrtev. Napad s slovarjem torej predpostavlja,

da si je uporabnik izbral polnopomensko geslo. Na spletu imamo na voljo veliko že narejenih slovarjev, bolj obširni so ponavadi veliki več gigabajtov. Če se sami lotevamo izdelave slovarjev, lahko kot osnovo uporabimo prave spletne slovarje, ki služijo jezikoslovcem (slovenski napadalci bi lahko uporabili SSKJ [38]).

Ko izpolnimo celotno formo pravilno, se v podatkovni bazi ustvari deaktiviran račun in uporabnik ga mora aktivirati, preden ga lahko začne uporabljati. Aktivacijska koda je poslana na elektronsko pošto novoustvarjenega profila kupca v obliki URL naslova. To je narejeno zato, da je težje ustvarjati izmišljene profile. Možno je namreč napisati program, ki avtomatsko izpolnjuje forme in jih pošilja, kar povzroči, da je baza spletne trgovine napolnjena s fiktivnimi osebami. Če bi kdo to želel narediti, bi moral napisati dodatni programski modul, ki bi ustvarjal elektronske poštne predale in potrjeval prijave, kar bi otežilo delo. Na spletu sicer obstajajo portali, ki omogočajo dostop do računov elektronskih pošt, ki jih ni treba ustvariti. Enostavno lahko dostopamo do njih, ker so last skupnosti in jih uporabljamo (na primer mailinator.com [39]). Vendar takih portalov trenutno ni tako veliko. Zato je večino njih možno navesti na črni listi v naši spletni trgovini, kar spet zmanjša verjetnost možne opisane težave.

Ko po registraciji in aktivaciji obiščemo stran s prijavnim obrazcem, moramo biti ponovno takoj preusmerjeni na HTTPS (preden oddamo uporabniško ime in geslo), drugače prijava ni varna. Kupec lahko vse podatke, ki jih je oddal portalu ob registraciji, razen uporabniškega imena in elektronskega naslova, kasneje tudi spreminja.

4.6.4 Ocenjevanje izdelkov

Prijavljeni uporabniki lahko le po enkrat ocenjujejo določen izdelek, medtem ko lahko neprijavljeni obiskovalci ocene izdelkov le gledajo. Nekatere spletne trgovine imajo sicer ocenjevanje omogočeno tudi za neprijavljene uporabnike, vendar to lahko povzroča probleme z relevantnostjo ocen. Tudi če hranimo IP naslove vseh, ki so glasovali, je sistem preveč odprt. Če na primer dve podjetji prodajata konkurenčne izdelke na isti spletni trgovini, je lahko ocena izdelka konkurenčna prednost, kar pomeni, da je v interesu obeh oceno zvišati. Poznamo primere [40], ko so se podjetja posluževala zlonamernih tehnik zviševanja števila glasov s t.i. "ugrabitvijo klikov"(angl. clickjacking) in "ugrabitvijo všečkov"(angl. likejacking, značilen predvsem za Facebook). Če lahko glasujejo le prijavljeni uporabniki, s

tem omejimo možnosti zlorabe, vendar posledično dobivamo manj glasov. Uporabnikom lahko ponudimo več vrst lestvic za ocenjevanje. Orginalna Likertova lestvica [41] je definirana tako, da se lahko odločimo za ocene od ena do pet, pri čemer ocena pet pomeni "Zelo mi je všeč" in ocena ena njeno popolno nasprotje. Poznamo tudi večtočkovne lestvice (sedem, devet, enajst), vsem pa je skupno, da imajo v sredini nevtralno točko, torej mora biti število točk liho. Za našo trgovino smo izbrali pet stopenjsko lestvico, ki je predstavljena z zvezdicami, ki so bele in se obarvajo črno, če gremo čeznje z miškinim kazalcem. Ob kliku na izbrano število zvezdic, se glas pošlje in zabeleži v podatkovni bazi. Vsak uporabnik ima tudi dostop do podatka, koliko ocen je posamezen artikel prejel in kolikšno je povprečje teh ocen.

4.6.5 Košarica in oddaja naročila

Košarice so v mnogih spletnih trgovinah implementirane tako, da se podatki o produktih v košarici hranijo v piškotku. Naša implementacija ima drugo možno rešitev in sicer, da se podatki o košarici shranijo kar v podatkovni bazi. Res je, da s tem zaledni sistem bolj obremenimo, vendar na ta način lahko isti nakup izvajamo z večih računalnikov (nadaljujemo lahko drugje) in se nam ni treba bati, da bo piškotek pretekel, ali da bi ga drug uporabnik našega računalnika izbrisal. Na strani sem implementiral standardno košarico, v katero lahko dodajamo produkte, ki jih imamo namen kupiti. Če si premislimo, lahko produkte izbrišemo iz košarice. Ko je košarica napolnjena, gremo "na blagajno", kjer se izpišejo vsi podatki o nakupu in lahko oddamo naročilo. Po oddaji naročila se košarica popolnoma izprazni. Posameznik lahko vsa svoja oddana naročila pregleduje tudi kasneje. Dostopa lahko do podatka o trenutnem stanju naročila, sistem pa beleži vso zgodovino o tem, kdaj je naročilo prišlo v kakšen del procesa. To je bolj podrobno opisano v nadaljevanju.

4.7 Izdelava modula za prodajalce

4.7.1 Pregled in upravljanje naročil

Prijavno okno za prodajalce se nahaja na isti strani, kot za administratorja. Ker nobena povezava iz splošnega modula za kupce ne kaže na prijavno stran, morajo

prodajalci in administrator v URL sami dopisati "adminLogin.php". Na ta način je to prijavno okno nekoliko težje dostopno široki množici nakupovalcev. Ko se prijavimo v modul za prodajalce, moramo ob prijavi strežniku predložiti veljaven certifikat. Prodajalec ima možnost upravljanja z naročili kupcev. Naročila so razdeljena v štiri razrede:

- Nepregledana.
- Odobrena.
- Zavrnjena.
- Stornirana.

Prodajalčeva naloga je, da nepregledana naročila razdeli v odobrena naročila ali zavrnjena naročila. Če kupec poslan izdelek kasneje zavrne, ga prodajalec uvrsti med stornirana naročila. Glede na to, kaj se zgodi z naročilom, se spremeni zaloga produktov, ki nastopajo v njem. Če je na primer naročilo odobreno, se zaloga odšteje, če naročilo kasneje zavrnemo, se zaloga ponovno prišteje. Po odobritvi mora prodajalec spletne trgovine poskrbeti, da po pošti pošlje izdelke. Sistem za preverjanje, ali je kupec naročilo plačal vnaprej s kreditno kartico, ni del te naloge, zato se predpostavlja, da gre za plačevanje po povzetju. Vsaka sprememba, ki jo prodajalec naredi z naročilom, se časovno in pomensko zabeleži, tako da je možno pregledovati zgodovino vseh sprememb. Da je poslovanje čim bolj transparentno, zgodovino lahko pregleduje tudi kupec za vsako svoje oddano naročilo posebej.

4.7.2 Upravljanje s produkti in kupci

Tudi prodajalec lahko (na enak način kot administrator) upravlja s produkti. Funkcionalnost, ki je značilna samo za prodajalca, je upravljanje s kupci. Gre za klasične operacije dodajanja in spreminjanja vseh atributov nekega profila, pri čemer lahko določenega uporabnika iz različnih razlogov deaktiviramo. Prodajalec seveda lahko upravlja tudi s svojimi atributi.

4.8 Oblikovna plat spletne trgovine

DRY (Don't Repeat Yourself) je princip programiranja, pri katerem kodo, ki jo večkrat uporabljamo, zapišemo na eno mesto. Na mesta, kjer se mora pojaviti, damo le referenco. S tem se izognemo ponavljanju. Tako programiranje nam omogoča CSS, ki je pravzaprav edina možna izbira za definiranje slogov aplikacij, katerih struktura bazira na HTML. Pri izbiranju barv sem kot ozadje, ki predstavlja največji delež prostora, vzel svetlo modro. Ozadje ni preveč nasičeno zato, da spletna trgovina ne izstopa preveč agresivno. Gumbi morajo zaradi tega, da jih lažje opazimo, izstopati. Zato sem zanje izbral rdečo in zeleno barvo. Trudil sem se, da stran izgleda čim bolj uravnoteženo, zato sem gradnike postavljal simetrično glede na vertikalno sredino. Prostor sem ponavadi razdelil s senčenjem in s črnimi črtami. Pri izdelavi prehodov barv, senčenja in oblike robov, sem si pomagal s spletnimi WYSIWYG (What You See Is What You Get) generatorji. Ti omogočajo hitrejše prikazovanje rezultatov dela in generirajo kodo glede na to, kako oblikovno spreminjamo objekt dela. Ker je za predstavitev spletne trgovine potrebno le-to napolniti s produkti, sem se odločil, da bom vanjo dal slike sodobnih pametnih telefonov in opise le-teh.

Poglavje 5

Rezultati

5.1 Testiranje aplikacije z OWASP ZAP

5.1.1 Način testiranja

Potrebno je omeniti, da sem aplikacijo med samim razvojem večkrat testiral in sproti odpravljal napake. V nadaljevanju je predstavljena zadnja verzija, ki ima tako najmanj odkritih napak.

Pri spletni trgovini sem najprej nastavil možnost, da klientu ob prijavi v zaledni sistem ni potrebno predložiti veljavnega certifikata. To naredi aplikacijo bolj ranljivo in ZAP najde več napak v kodi. Za napredno testiranje aplikacije se v brskalnik Mozilla Firefox najprej naloži vtičnik "Plug-n-Hack", ki ZAP omogoča beleženje vseh zahtevkov, ki jih v omenjenem brskalniku pošljemo strežniku. Ker ima testirana aplikacija lahko povezave na druge spletne strani, ki niso v naši lasti, se izključi povezava do spleta. ZAP bi v nasprotnem primeru napadal tudi te spletne strani. Testiranje je bilo razdeljeno na dva dela. V prvem delu sem testiral strani, do katerih je možno dostopati prek HTTP zahtevkov (neprijavljeni kupci), v drugem delu pa strani, do katerih je mogoče dostopati le preko HTTPS zahtevkov (prijavljeni kupci, celoten zaledni sistem). Struktura posameznega testiranja je sestavljena iz treh delov:

1. Zbiranje informacij s pajkom (angl. Spider).
2. Ročno pošiljanje zahtevkov preko brskalnika.

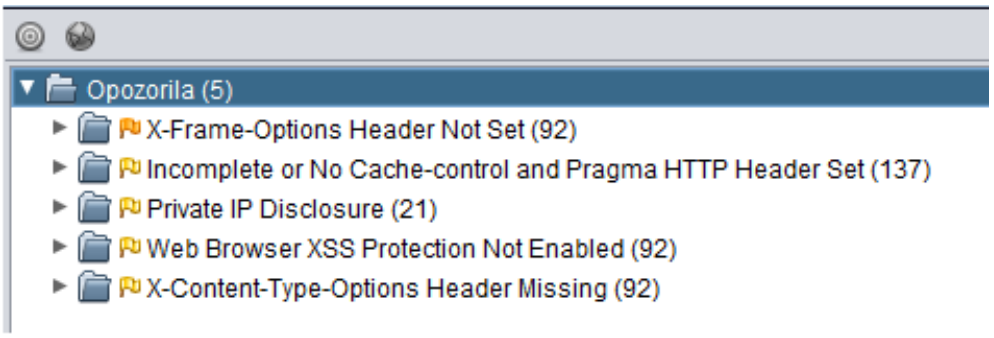
3. Skeniranje (angl. Active Scan).

Zbiranje informacij s pajkom nam rekurzivno preišče vse hiperpovezave, ki nastopajo na spletni strani. S tem dobi splošen pregled nad strukturo aplikacije. Ročno pošiljanje zahtevkov v ZAP velja za napreden način, s katerim programu definiramo vse možne funkcionalnosti aplikacije. Z brskalnikom moramo torej uporabiti vse funkcionalnosti, ki smo jih implementirali. Če tega ne naredimo, se zna zgoditi, da ZAP nekaterih napadov ne bo testiral. Nastaviti moramo tudi preko katerih vektorjev bomo napadali. Omogočil sem vse možnosti:

- URL poizvedbe (angl. URL Query String) - napadi preko GET zahtevkov.
- Napadi preko POST zahtevkov (angl. POST Data).
- Testiranje poti preko URL (angl. URL Path).
- Spreminjanje HTTP zaglavij (angl. HTTP Headers).
- Preverjanje piškotkov (angl. Cookie Data).

5.1.2 Rezultati testiranja

ZAP je spletni trgovini prek HTTP povezave poslal 23042 zahtevkov. Prek HTTPS povezave je bilo dodatno poslanih 101182 zahtevkov. Vsak zahtevek je napad na aplikacijo. Rezultat testiranja z ZAP so opozorila, ki kažejo na potencialne ranljivosti aplikacije. Včasih se zgodi, da opozarja na napake, ki se v širšem konteksu izkažejo, da to pravzaprav niso, zato moramo natančno preveriti vsako opozorilo posebej. ZAP opozorila uvršča v tri skupine, glede na to, koliko je odkrita napaka nevarna (malo, srednje in zelo nevarna). Če posamezno opozorilo bolj natančno pogledamo, se nam pokažejo vsi zahtevki, iz katerih ZAP sklepa, da je aplikacija ranljiva. Oba testa (HTTP in HTTPS) sem izvedel zaporedno in opozorila obeh so zbrana v enem poročilu, ki je prikazan na sliki 5.1.



Slika 5.1: Opozorila programa OWASP ZAP po testiranju.

Opozoril, ki bi spadala med zelo nevarna ni, eno opozorilo je srednje nevarno in štiri so malo nevarna. Če najprej podrobneje pogledamo srednje nevarno opozorilo *X-Frame-Options Header Not Set*, ki vsebuje 92 zahtevkov, vidimo, da so znotraj njega vsi navedeni zahtevki tipa GET. Poleg te skupne lastnosti za vsak zahtevek posebej velja, da gre pri njem izključno za enega izmed naslednjih dostopov: dostop do datoteke tipa **.css*, dostop do datoteke tipa **.js* ali dostop do mape, pri katerem ni definirana datoteka. To opozorilo pomeni, da stran omogoča vgrajevanje okvirov (npr. iframe) v spletno stran, kar omogoča izvedbo clickjacking napada. A ker zgolj v tipe datotek **.css* ali **.js* (brez strežnikove logike) in v samo mapo ne moremo vključevati okvirov, je prvo opozorilo nerelevantno.

Če pogledamo številko 92, ki se pojavlja pri ravnokar opisanem opozorilu, vidimo, da se enaka številka pojavlja tudi pri opozorilih *Web Browser XSS Protection Not Enabled* in *X-Content-Type-Options Header Missing*. Opazimo, da gre za iste zahtevke. Podobno, kot v te tipe datotek (**.css*, **.js*) ni mogoče vključevati okvirov, vanje tudi (neposredno brez logike strežnika) ni mogoče vključevati JavaScript kode in različnih parametrov HTTP odzivov. Datoteke lahko samo beremo, strežnik pa nas zavrne (izključen *Directory listing*) ob dostopanju do map. Zato tudi ti dve opozorili nista relevantni za dotično aplikacijo.

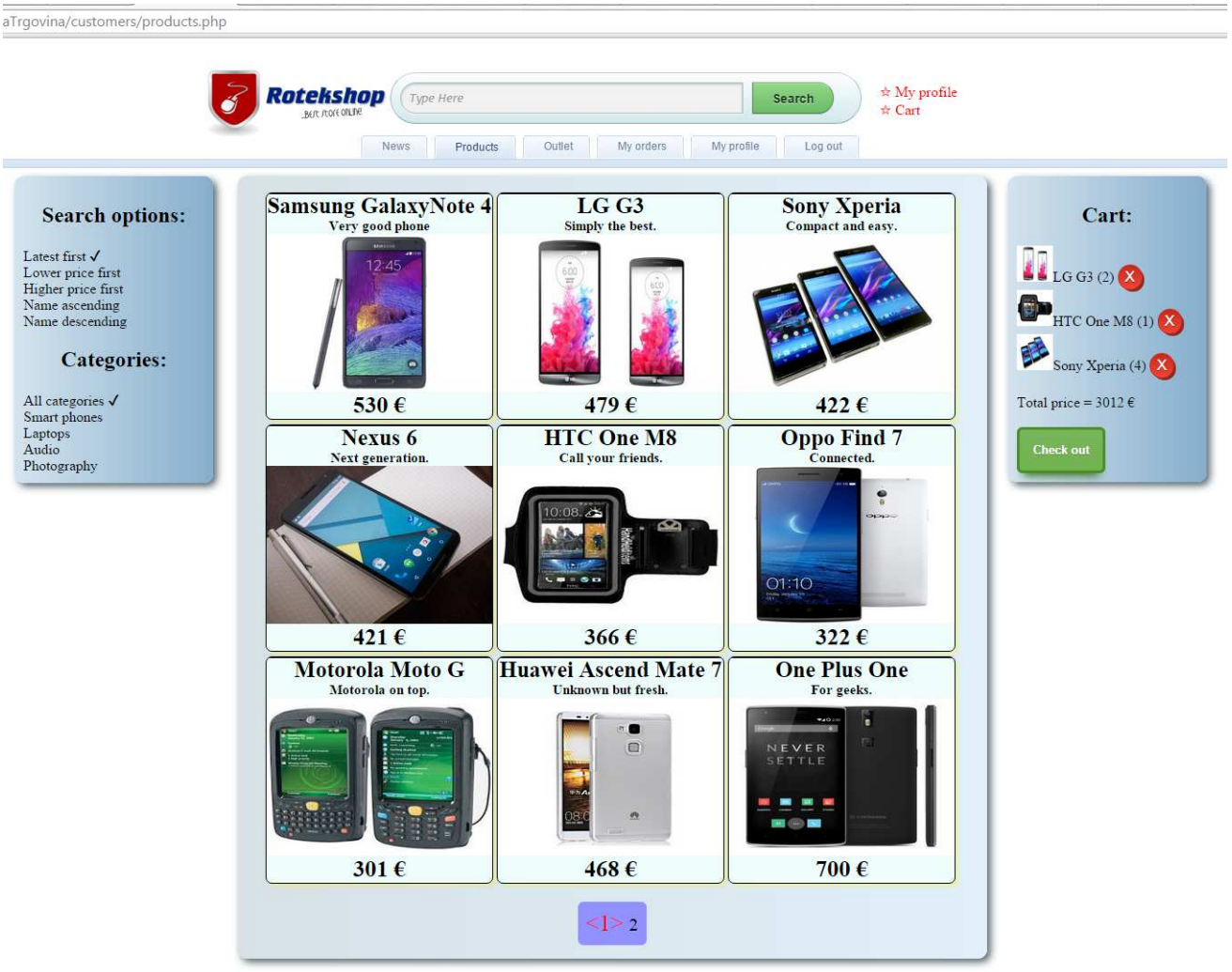
Ostaneta nam še dve opozorili. *Incomplete or No Cache-control and Pragma HTTP Header Set* priporoča, da moramo, kadar je le mogoče, HTTP zaglavje (angl. header) nastaviti kot je opisano v nadaljevanju. Parameter *Cache-control* se nastavi na: "no-cache, no-store, must-revalidate"; medtem ko se *Pragma* nastavi na "no-cache". Vsebina HTTP zaglavij, ki jih pošilja strežnik, je tudi nastavljena po priporočilih (spomnimo se 4. poglavja Implementacija), zato je opozorilo upoštevano.

Zadnje opozorilo *Private IP Disclosure* je pomota programa ZAP, saj je odsek zaporedja številk, ki opisujejo postavitev slike (...10-116-115-47...) prepoznal kot privatno omrežje (oblike 10.0.0.0).

Zdaj lahko rečemo, da smo odpravili vse pomanjkljivosti, ki jih je zaznal ZAP. Če ponovno vključimo še zahtevo, da mora klient med prijavo predložiti certifikat, je aplikacija še bolj varna.

5.1.3 Prikaz funkcionalnosti v obliki zaslonskih slik

Na naslednjih slikah je prikazan grafični vmesnik za uporabo realiziranih funkcionalnosti, ki sem jih opisoval v prejšnjem poglavju. Ker aplikacijo želim ponuditi čim širšemu krogu uporabnikov, je prva različica razvita v angleškem jeziku. V prihodnosti nameravam dodati še podporo drugim jezikom. Na sliki 5.2 vidimo na skrajni levi iskalnik po kategorijah, zgoraj pa iskalnik po vnešenem nizu ter meni z raznimi zavihki. Na skrajni desni je košarica.

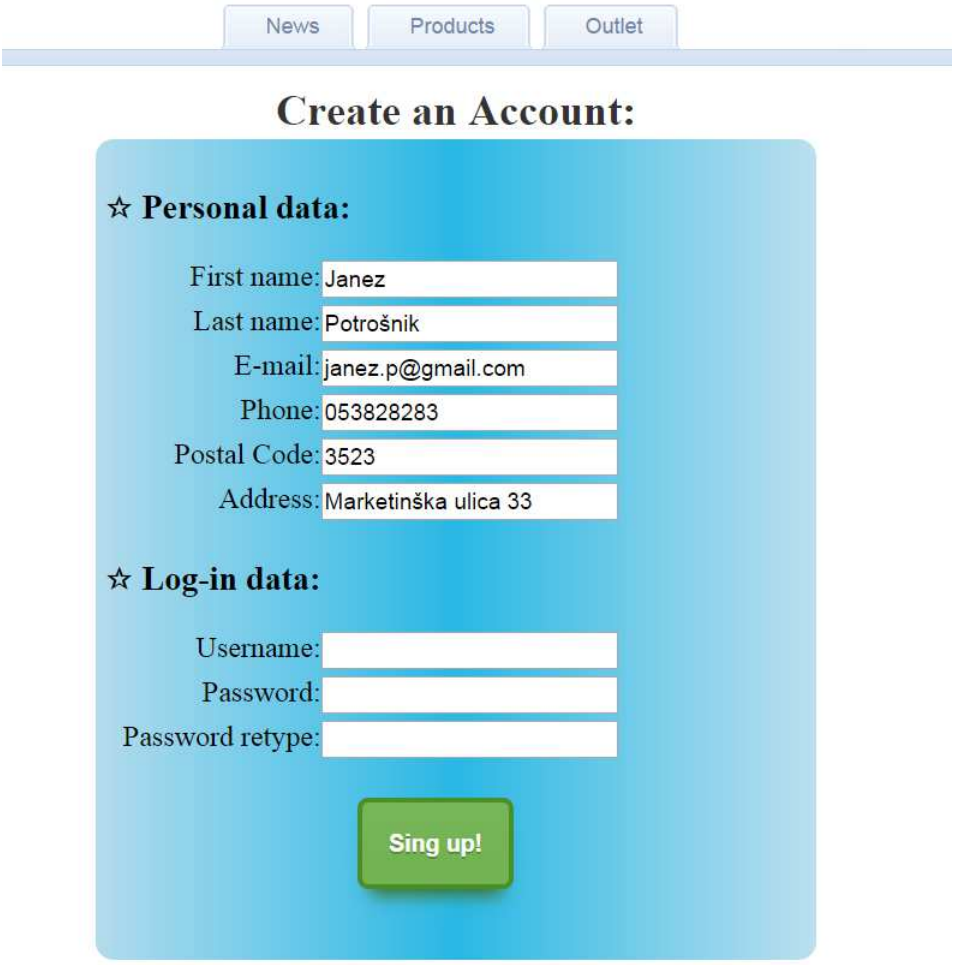


Slika 5.2: Pregledovanje in iskanje produktov.

Če izberemo nek produkt, se nam prikažejo posamezne podrobnosti le-tega, kot je prikazano na sliki 5.3. V galeriji lahko pregledujemo slike izdelka, ga ocenjujemo in damo v košarico. Ko izberemo sliko, se slika poveča.

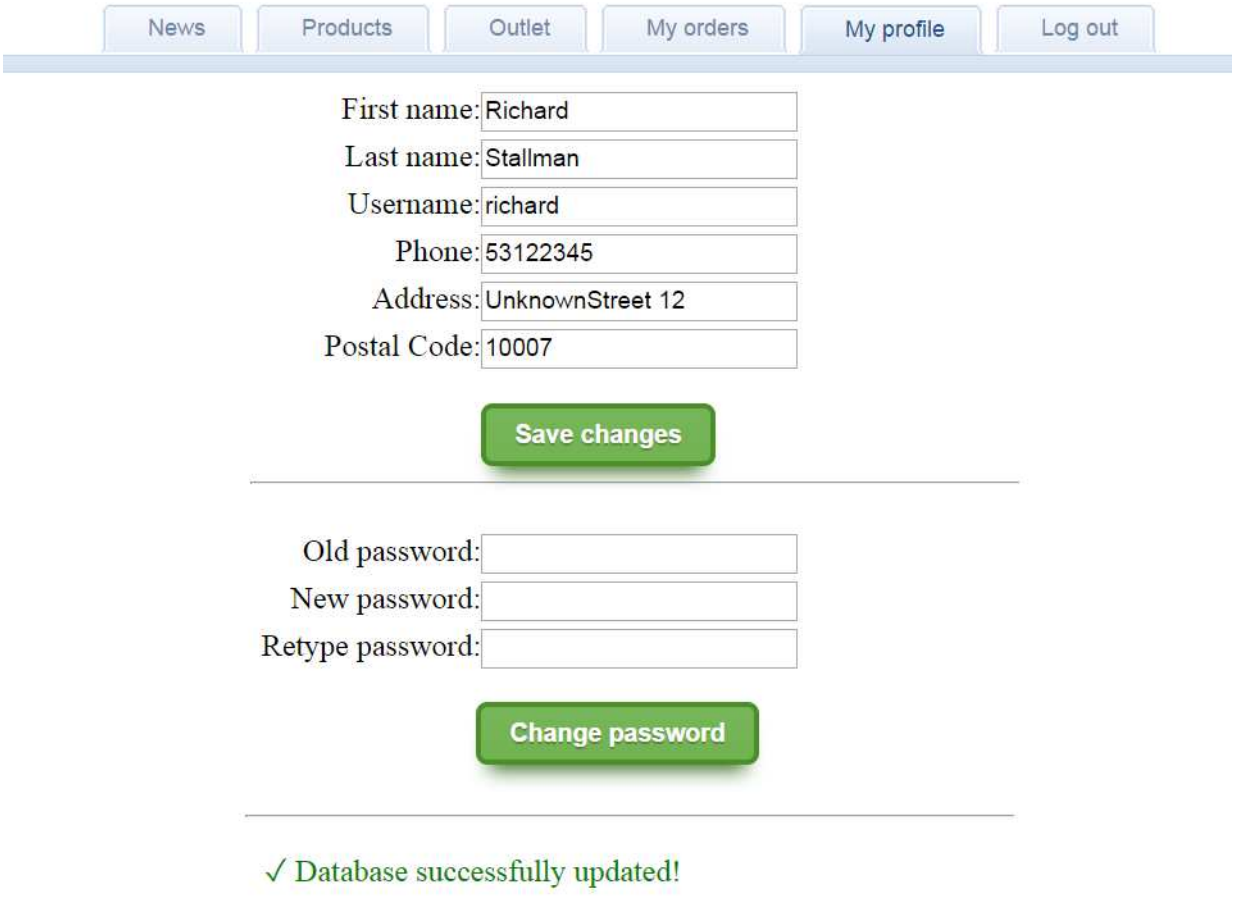
Slika 5.3: Pregled podrobnosti izbranega produkta.

Na sliki 5.4 je prikazan obrazec, ki ga je potrebno izpolniti, ko ustvarjamo nov profil kupca. Če ob vnosu naredimo kakšno napako in v polje obrazca vnesemo neustrezen niz znakov, nas sistem na napako opozori.



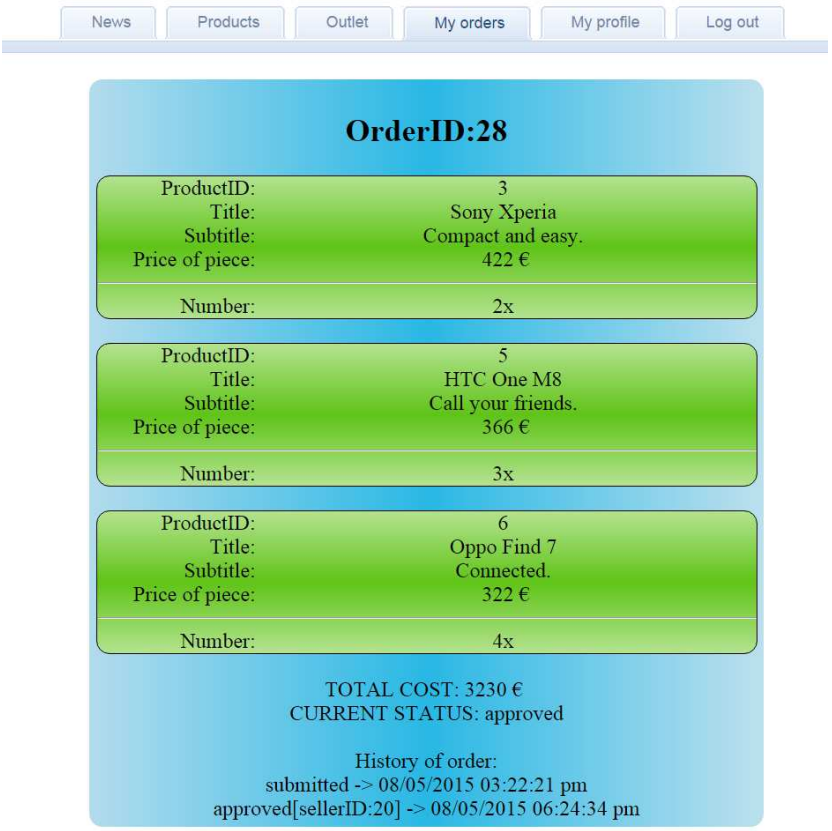
Slika 5.4: Forma za ustvarjanje novega profila kupca.

Vsak uporabnik lahko spreminja podatke svojega profila. To je prikazano na sliki 5.5. Te podatke spletna trgovina uporabi za pošiljanje produkta.



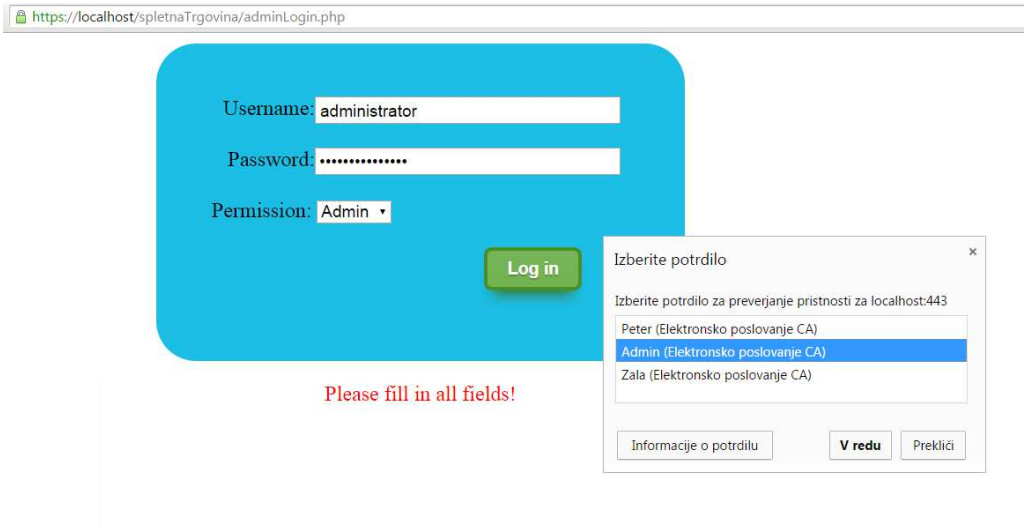
Slika 5.5: Spreminjanje podatkov profila za kupca.

Slika 5.6 prikazuje, kako lahko posamezen kupec pregleduje zgodovino svojih oddanih naročil.



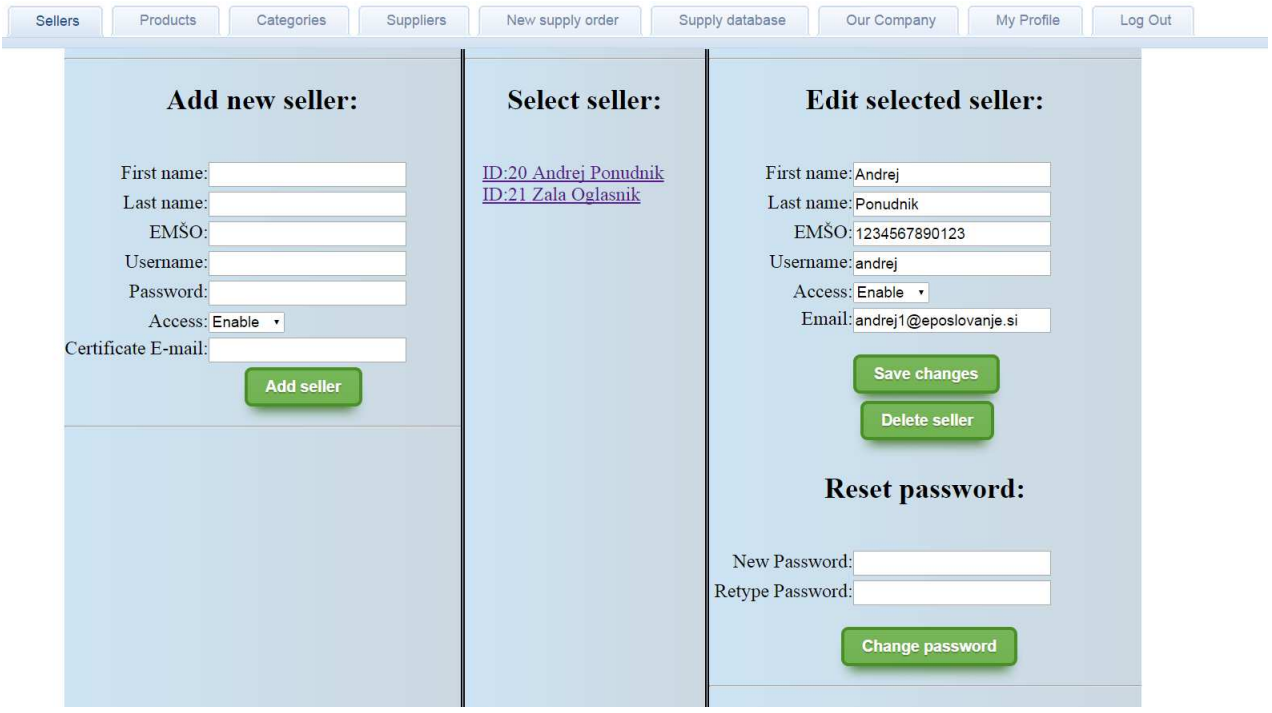
Slika 5.6: Pregledovanje preteklih naročil prijavljenega kupca.

Na sliki 5.7 vidimo prijavni obrazec za administratorja in prodajalce. Zahtevana je predložitev certifikata.



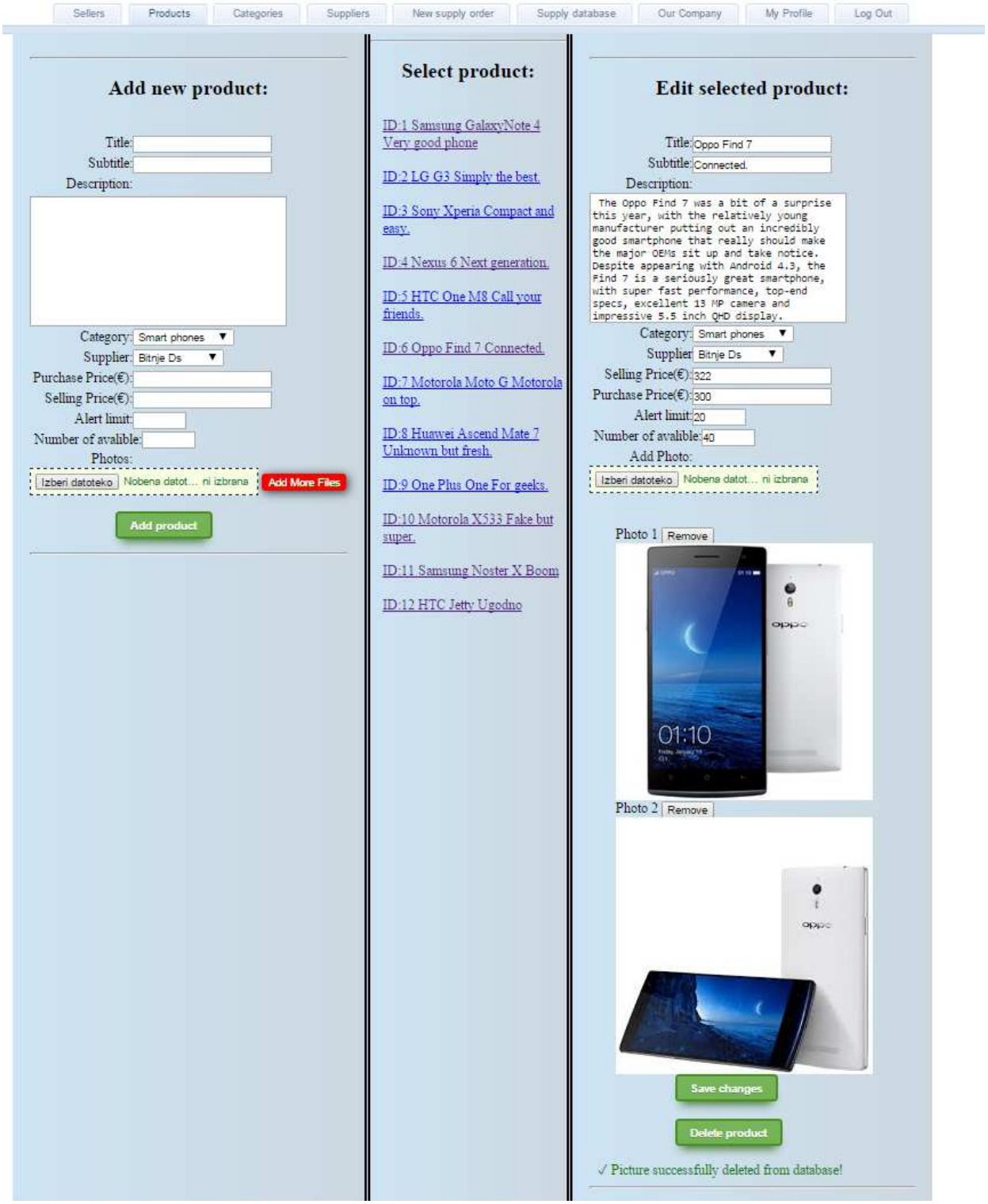
Slika 5.7: Prijava v zaledni sistem administratorja ali prodajalca, kjer je obvezna predložitev certifikatov.

Slika 5.8 prikazuje zaledni sistem administratorja. Bolj natančno, gre za funkcionalnost upravljanja s profili prodajalcev.



Slika 5.8: Dodajanje, spreminjanje ali brisanje profila prodajalcev v spletni trgovini za administratorja.

Na sliki 5.9 vidimo funkcionalnost za urejanje produktov. Posameznemu produktu definiramo dobavitelja, nabavno in prodajno ceno ter opis. Dodamo lahko tudi poljubno število slik.

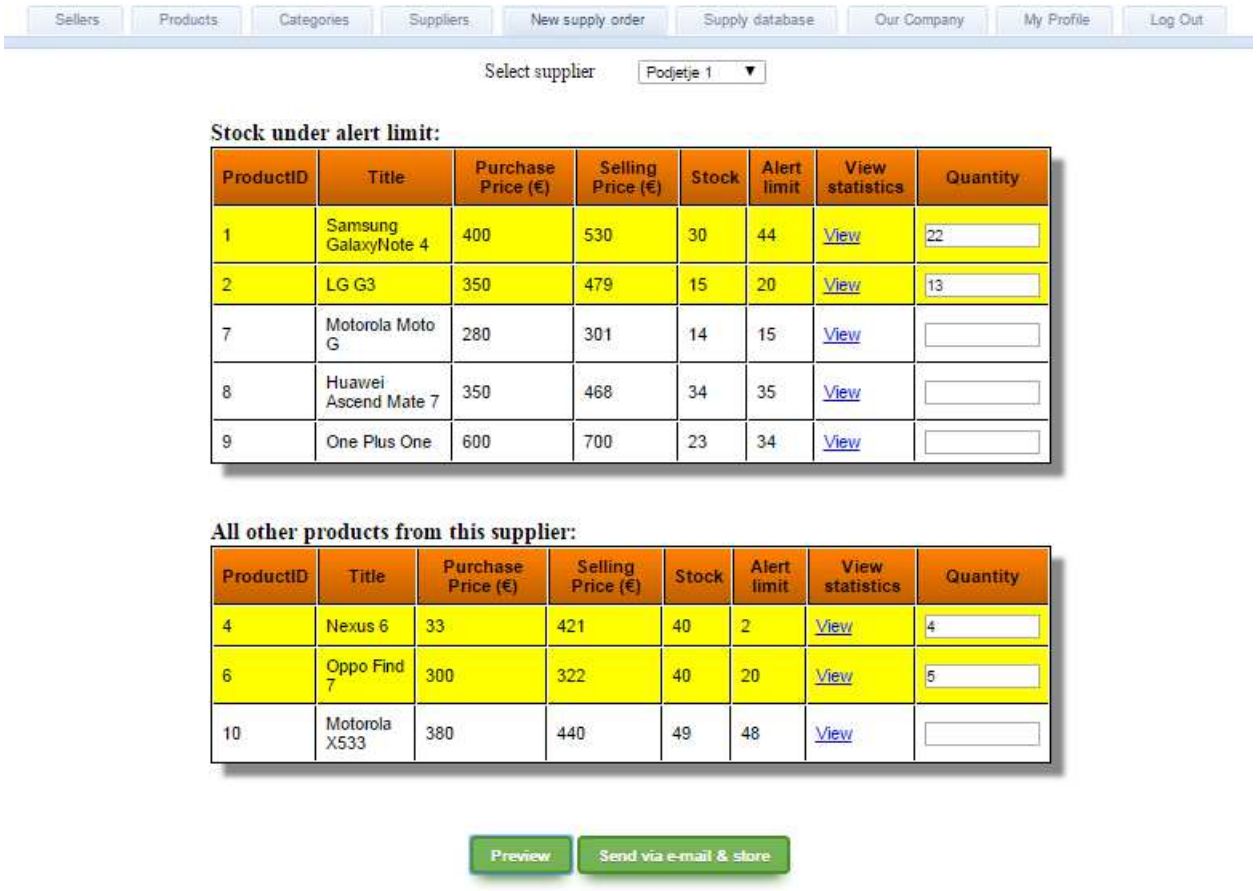


Slika 5.9: Dodajanje, spreminjanje ali brisanje produktov v spletni trgovini (funkcionalnost administratorja).

Na sliki 5.10 je prikazano urejanje kategorij.

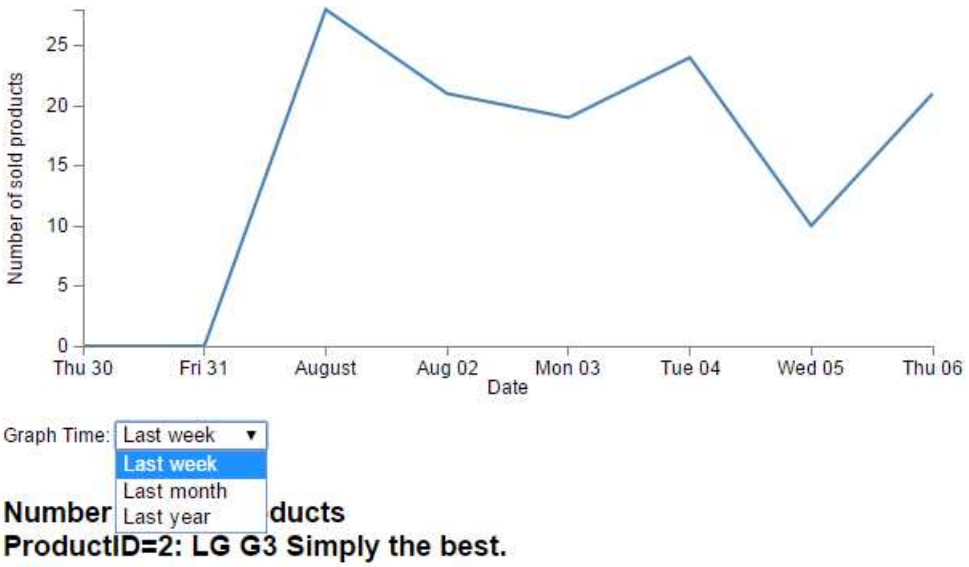
Slika 5.10: Dodajanje, spreminjanje ali brisanje kategorije v spletni trgovini (funkcionalnost administratorja).

Poseben modul je namenjen naročanju pri dobaviteljih. Slika 5.11 prikazuje izgled vmesnika za naročanje pri dobaviteljih. Z enojnim klikom preprosto izberemo produkte in v obrazec vnesemo število produktov, ki jih želimo naročiti.



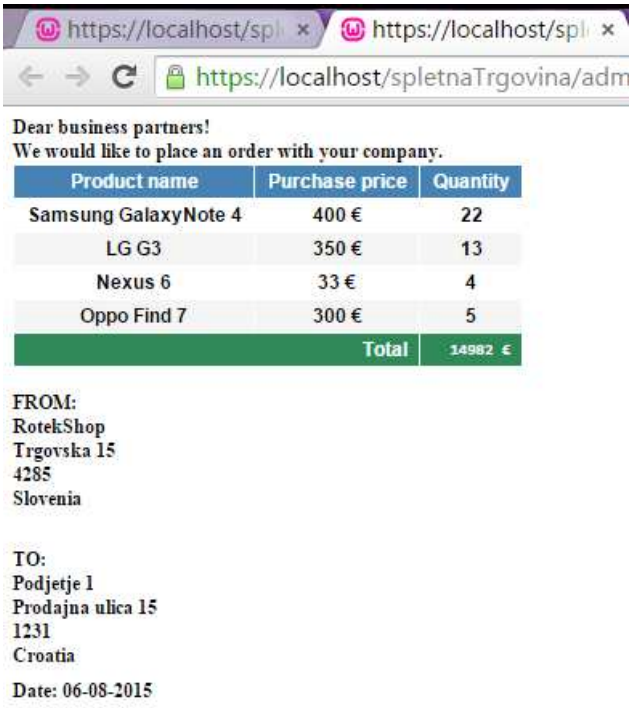
Slika 5.11: Naročanje novih produktov (funkcionalnost administratorja).

Slika 5.12 prikazuje graf, ki podaja statistiko prodaje posameznega izdelka.



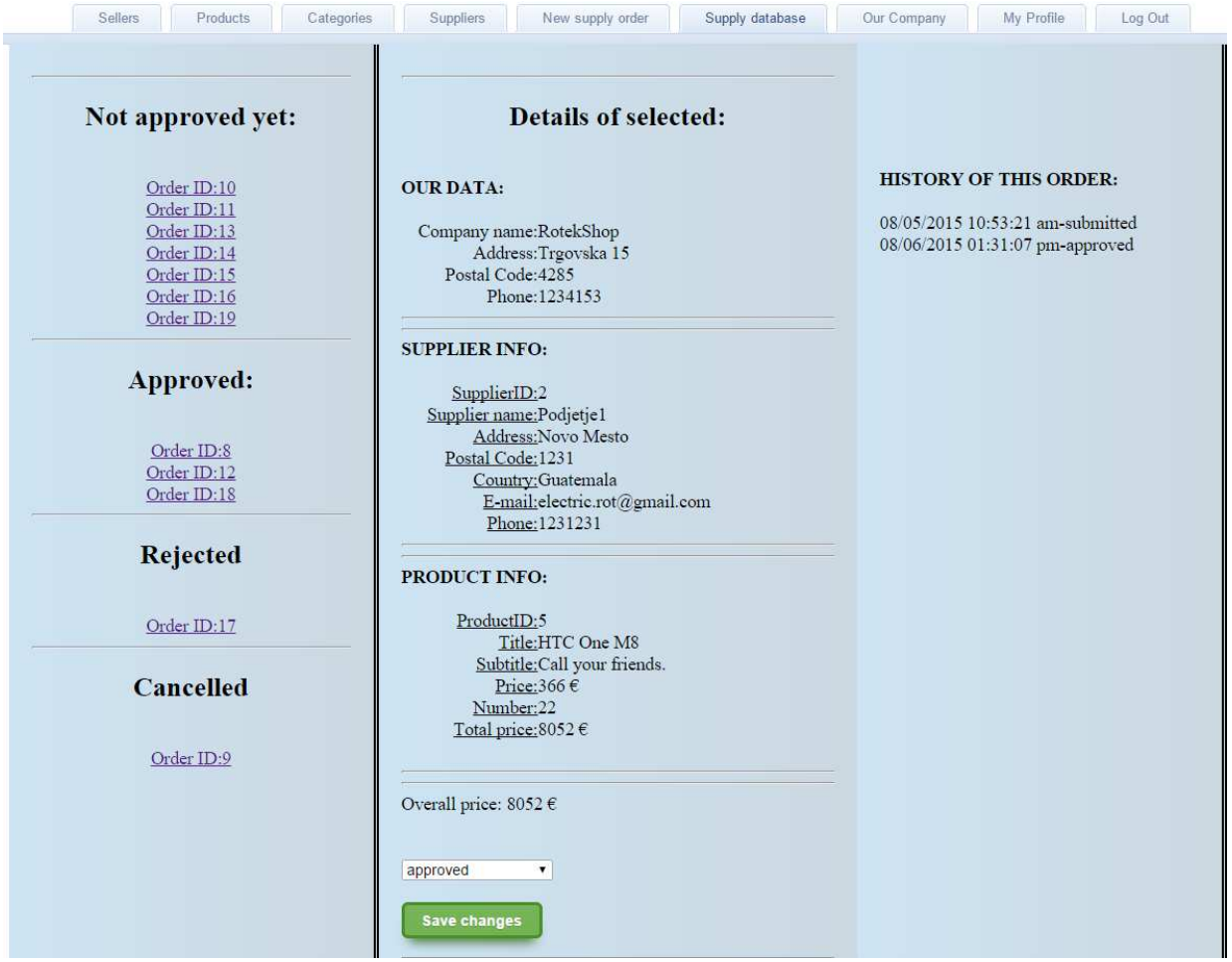
Slika 5.12: Prikaz statistike prodaje izdelkov za zadnji teden, mesec in leto.

Slika 5.13 prikazuje predogled naročila izdelkov pri posameznem dobavitelju.



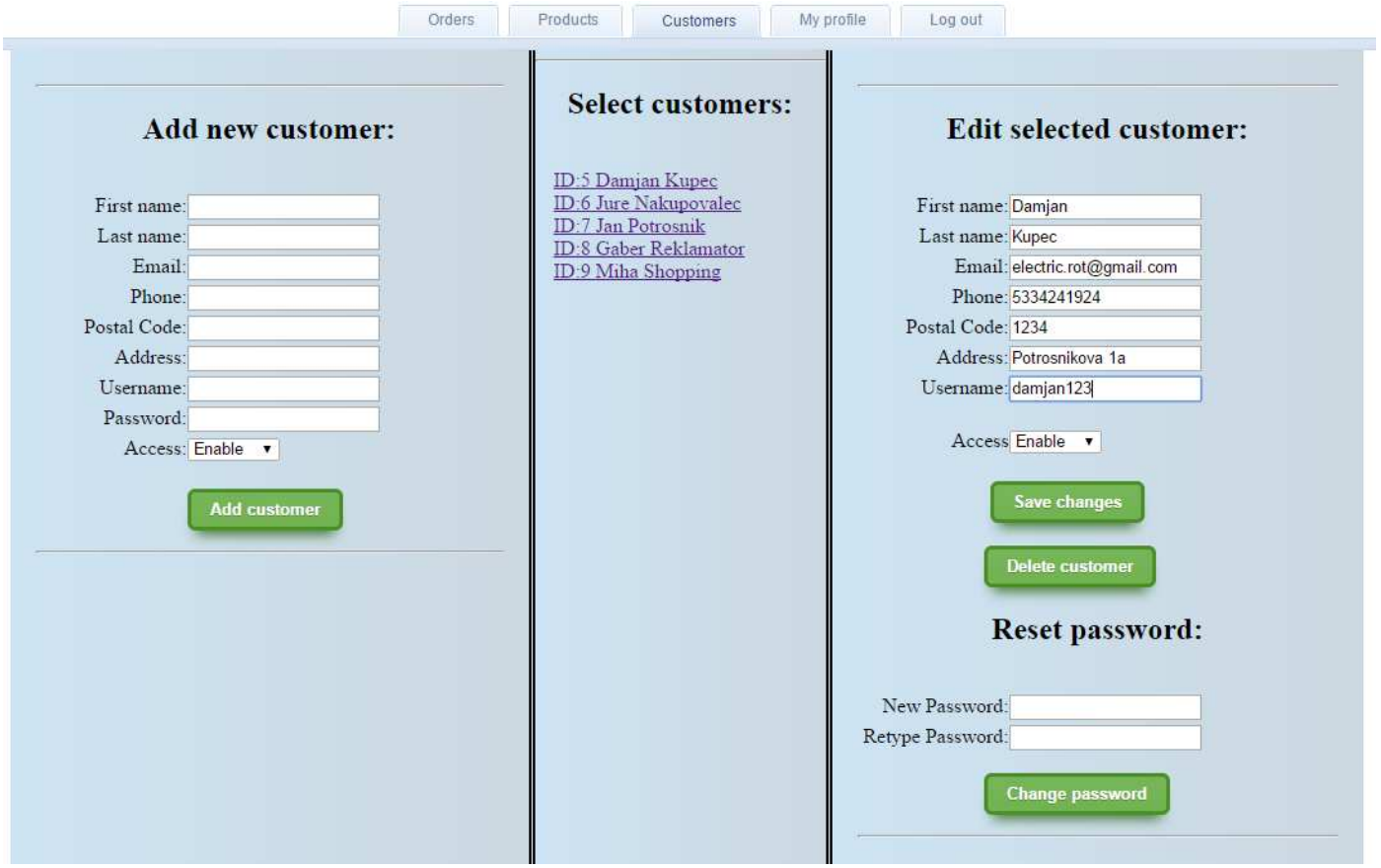
Slika 5.13: Predogled naročila pred oddajo.

Na sliki 5.14 vidimo pregled preteklih naročil dobaviteljem. Na podoben način je realizirano tudi sprejemanje naročil pri prodajalcih.



Slika 5.14: Pregled oddanih naročil, ki omogoča njihovo razporejevanje v kategorije in pogled v detajle.

Slika 5.15 prikazuje funkcionalnost, ki jo imajo samo prodajalci. Gre za upravljanje s profili kupcev.



Slika 5.15: Urejanje atributov kupca, ki je del modula za prodajalce.

Poglavje 6

Sklepi

6.1 Sklepne ugotovitve

Kot je bilo povedano v uvodu, sta bila cilja te naloge dva. Prvi cilj je bil razvoj funkcionalnosti spletne trgovine. Uspešno so bili implementirani vsi trije moduli - modul za kupce, prodajalce in administratorja. Podatkovna baza je v želeni tretji normalni obliki. Samopodpisani certifikati so bili uspešno izdelani in sistem za avtentikacijo administratorja ter prodajalcev deluje, kot je bilo načrtovano. Kupci lahko na preprost način naročajo produkte, ki jih plačajo po povzetju. Programsko kodo bi teoretično lahko uporabljali trgovci, ne glede na to, kakšne produkte želijo prodajati preko spleta. Ciljna skupina uporabnikov je zato relativno velika. Ocenjujem, da je upravljanje s slikovnim gradivom produktov narejeno na zadovoljivi ravni. Realiziran je tudi modul za naročanje pri proizvajalcih, ki vsebuje preprost priporočilni sistem. Projekt je glede funkcionalnosti zelo razširljiv, saj ga lahko dopolnimo z mnogimi novimi moduli. Uporabljene tehnologije so se pri tem projektu, glede na zahtevano hitrost procesiranja spletnih strani, izkazale za pravilno izbrane, saj je procesiranje posamezne strani zelo hitro. Nimamo težav s čakanjem na odziv strežnika, kar je za potencialne kupce dobro.

Zavedam se, da golo število vrstic kode ne pove veliko o projektu, vseeno pa navajam nekaj podatkov o končnem izdelku. Osnutek projekta je začel nastajati novembra 2014 v okviru seminarske naloge pri predmetu Elektronsko poslovanje. Od tod izhaja tudi ideja za tematiko te diplomske naloge. V izdelku je po principu DRY v šestdesetih datotekah napisanih 5511 vrstic efektivne PHP kode, v devetih

CSS datotekah 1122 vrstic kode, ter približno 250 vrstic JavaScript kode (brez knjižnic).

Drugi cilj te naloge pa je bil zagotavljanje varnosti. Kot je razvidno iz rezultatov testa, ZAP ni odkril nevarnosti za injekcije, XSS napade in CSRF napade. Podatki v podatkovni bazi so glede na smernice organizacije OWASP pravilno shranjeni (zgoščevalne funkcije za gesla). Nevarnosti z nepravilnim upravljanjem avtentikacije in sejami so razrešene na zadovoljivi ravni. Do preusmeritve na HTTPS povezave prihaja na pravih mestih, sistem dovoli le dovolj kompleksna gesla in ID seje je shranjen v spremenljivki seje. Konfiguracija strežnika je nastavljena po priporočilih tako, da je izključena možnost prikazovanja vsebine map na strežniku. Tudi javljanje napak v PHP je izključeno. Vseh teh napak ZAP pričakovano ni zaznal. Ob vsem naštetem mislim, da sem bil pri delu uspešen.

Na tem mestu, bi bilo smiselno pogledati še, katerim vrstam napadov v tej nalogi nisem posvetil posebne pozornosti. Bralec lahko najde podrobnejše opise v OWASP dokumentaciji [5]. To so napadi:

- Zavrnitev storitve ali Denial of Service (DoS).
- ReDoS (Regular expression DoS) - napad temelji na dejstvu, da se nekatere regularne izraze dolgo preverja.
- Prečkanje poti (angl. Path traversal) - pri ZAP se je vseeno izkazalo, da aplikacija ni občutljiva nanj.
- Zastrupljanje predpomnilnika (angl. Cache-poisoning) - izkazalo se je, da napad v tem projektu prepreči priporočena konfiguracija HTTP odzivov, ki sem jo upošteval v poglavju 4.4 Nastavitev HTTP odzivov.
- Dvojno kodiranje (angl. Double encoding) - nevarnost je odpravila zaščita pred XSS napadi.

6.2 Možnosti za nadaljevanje dela

Možnosti, ki niso bile del diplomske naloge in jih vidim kot izboljšanje dosedanje aplikacije, so naslednje:

- Trgovina bi lahko podpirala več jezikov. To pomeni spreminjanje podatkovne baze v velikem obsegu in precejšnje prilagajanje obrazcev za dodajanje produktov.
- OWASP (poleg vseh implementiranih zaščit proti napadom) opisuje še mnogo napadov, ki so sicer manj pogosti, a prav tako nevarni. Dalo bi se torej implementirati več obrambnih mehanizmov.
- Za boljšo praktično uporabo bi bila koristna implementacija plačevanja preko interneta.
- Naslednja možna izboljšava je izrisovanje drugačnega grafičnega vmesnika na različnih napravah. To je do neke mere realizirano, vendar je pri tem možnih še veliko izboljšav. Lahko bi torej bolj natančno priredili izris za posamezne velikosti mobilnih telefonov in tabličnih računalnikov.
- Za boljšo podporo pri odločanju (naročanje od dobaviteljev), bi lahko s knjižnico D3.js izrisoval še drugačne grafe. Grafi bi na več načinov opisovali trende prodaje.
- V modul za nakupovanje bi lahko vključil še priporočilni sistem (osnovan na primer na matrični faktorizaciji).

Literatura

[1] Apache Software Foundation. Apache HTTP Server Project, 2015. URL http://httpd.apache.org.

[2] January 2015 web server survey, 2015. URL http://news.netcraft.com/archives/2015/01/15/january-2015-web-server-survey.html. [Online; dostopno 4-August-2015].

[3] OWASP ZAP, 2015. URL https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.

[4] Owasp zap reconnaissance – without permission!, 2015. URL http://resources.infosecinstitute.com/owasp-zap-reconnaissance-without-permission/. [Online; dostopno 4-August-2015].

[5] OWASP Top 10, 2015. URL http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

[6] Padriac Brady. Survive the deep end: Php security, 2015. URL http://phpsecurity.readthedocs.org/en/latest.

[7] StackOverflow Inc. StackOverflow, 2015. URL http://stackoverflow.com/.

[8] Matthias Gelbmann. Usage of programming languages for websites, 2015. URL http://w3techs.com/technologies/overview/programming_language/all.

[9] Adobe Systems Incorporated. Getting started with actionscript 3, 2015. URL http://www.adobe.com/devnet/actionscript/getting_started.html.

[10] Wikipedia. Ajax (programming) — wikipedia, the free encyclopedia, 2015. URL https://en.wikipedia.org/w/index.php?title=Ajax_(programming)&oldid=672524860. [Online; dostopno 7-August-2015].

[11] Inc. DropBox. What is dropbox?, 2015. URL https://www.dropbox.com/news/company-info. [Online; dostopno 7-August-2015].

[12] Google. Google drive, 2015. URL http://www.google.com/intl/sl_SI/drive/. [Online; dostopno 7-August-2015].

[13] Mozy homepage, 2015. URL https://mozy.ie/#slide-8. [Online; dostopno 7-August-2015].

[14] Amazon homepage, 2015. URL http://www.amazon.com/. [Online; dostopno 9-August-2015].

[15] eBay, 2015. URL http://www.ebay.com/. [Online; dostopno 9-August-2015].

[16] Online retailing: Britain, europe, us and canada 2015, 2015. URL http://www.retailresearch.org/onlineretailing.php. [Online; dostopno 7-August-2015].

[17] Send money, pay online or set up a merchant account, 2015. URL http://www.paypal.com/si/webapps/mpp/home. [Online; dostopno 7-August-2015].

[18] Mastercard homepage, 2015. URL http://www.mastercard.com/si/consumer/. [Online; dostopno 7-August-2015].

[19] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Consulted*, 1(2012):28, 2008.

[20] Googleads homepage, 2015. URL http://www.google.com/ads/. [Online; dostopno 7-August-2015].

[21] Wordpress homepage, 2015. URL https://wordpress.com/. [Online; dostopno 7-August-2015].

[22] Joomla homepage, 2015. URL http://www.joomla.org/. [Online; dostopno 7-August-2015].

[23] Martyn Williams. Sony apologizes, details PlayStation Newtork attack, 2011. URL http://www.csoonline.com/article/2128432/data-protection/sony-apologizes--details-playstation-network-attack.html.

[24] Christian Schneider. CSRF and Same-Origin XSS, 2012. URL http://http://www.christian-schneider.net/CsrfAndSameOriginXss.html.

[25] Wikipedia. Man-in-the-middle attack — wikipedia, the free encyclopedia, 2015. URL https://en.wikipedia.org/w/index.php?title=Man-in-the-middle_attack&oldid=672937121. [Online; dostopno 1-August-2015].

[26] Eric Butler. Firesheep, 2010. URL http://codebutler.com/firesheep.

[27] Php session ids - how are they generated, 2014. URL http://stackoverflow.com/questions/18937651/php-session-ids-how-are-they-generated. [Online; dostopno 14-August-2015].

[28] Jennifer Widom Hector Garcia-Molina, Jeffrey D. Ullman. *DATABASE SYSTEMS The Complete Book Second Edition*. "Pearson Education Inc. ", 2009.

[29] Mark Masse. *REST API design rulebook*. "O'Reilly Media, Inc.", 2011.

[30] Todd Garrison. Using Google Chrome for security testing, 2014. URL http://www.frameloss.org/2011/11/01/using-google-chrome-for-security-testing/.

[31] WhiteHat Security. Csrf Prevention in Java, 2012. URL http://http://www.christian-schneider.net/CsrfAndSameOriginXss.html.

[32] Denis Trček Iztok Starc. Elektronsko poslovanje, študijska skripta. 2014.

[33] Apache Software Foundation. Apache module mod ssl, 2015. URL http://httpd.apache.org/docs/2.2/mod/mod_ssl.html.

[34] Mike Bostock. Data Driven Documents, 2015. URL http://d3js.org/.

[35] Magnetic Marketing Corp. EasyRotator - Free jQuery Slider / Rotator Builder, 2015. URL http://www.dwuser.com/easyrotator/.

[36] The university of Texas at Austin. Keep Safe with Strong Passwords, 2015. URL http://www.utexas.edu/its/secure/articles/keep_safe_with_strong_passwords.php.

[37] Kali Linux community. Penetration Testing and Ethical Hacking Distribution, 2015. URL http://www.kali.org/.

[38] SAZU. Slovenski slovar knjižnega jezika, 2000. URL http://bos.zrc-sazu.si/sskj.html.

[39] ManyBrain. The Mailinator FAQ, 2013. URL https://mailinator.com/faq.html.

[40] BBC news. Facebook sues alleged clickjacking spammer sparking row, 2012. URL http://www.bbc.com/news/technology-16755434.

[41] Wikipedia. Likert scale — wikipedia, the free encyclopedia, 2015. URL https://en.wikipedia.org/w/index.php?title=Likert_scale&oldid=668657199. [Online; dostopno 4-August-2015].